凝聚名家技术典范·分享成功IT之路



循序渐进DB2(第2版)

——DBA系统管理、运维与应用案例



作序推荐

王阳
IBM全球副总裁
IBM中国开发中心总经理

胡世忠
IBM全球副总裁
IBM软件集团大中华区总经理

牛新庄 著

清华大学出版社

循序渐进 DB2(第 2 版)——DBA 系统管理、运维与应用案例

牛新庄　著

清华大学出版社

北　　京

内 容 简 介

DB2 数据库是 IBM 公司关系型数据库核心产品，在国内以及全球有着广泛的应用。针对 DB2 初学者，本书循序渐进地把 DB2 涉及的众多概念和知识介绍给大家。客户端连通性、实例、数据库、表空间和缓冲池、数据移动、备份恢复、SQL 基础知识、DB2 基本监控方法、运行数据库必须考虑的设置、DBA 日常维护以及数据库常用工具都是本书关注的重点。在介绍这些数据库对象和概念的同时，作者尽可能从 DBA 日常工作的角度探究 DB2 数据库常规维护工作。本书同时还就表、索引、序列、触发器等数据库对象从应用设计的角度进行了介绍。本书适合 DB2 的初学者、DB2 开发人员、准备参加 DB2 认证考试的读者以及 DB2 数据库管理人员学习和阅读。

图书在版编目(CIP)数据

循序渐进 DB2：DBA 系统管理、运维与应用案例/牛新庄 著；—2 版. —北京：清华大学出版社，2013.7
ISBN 978-7-302-32301-3

Ⅰ. ①循…　Ⅱ. ①牛…　Ⅲ. ①关系数据库系统　Ⅳ. ①TP311.138

中国版本图书馆 CIP 数据核字(2013)第 091910 号

责任编辑：王　军　李维杰
装帧设计：牛艳敏
责任校对：蔡　娟
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×230mm　　**印　　张：**38.25　　**字　　数：**788 千字
版　　次：2009 年 5 月第 1 版　2013 年 7 月第 2 版　　**印　　次：**2013 年 7 月第 1 次印刷
印　　数：1～4000
定　　价：80.00 元

产品编号：

序　一

新庄是 IBM 的老朋友，也是我的老朋友了。虽然我们的见面次数并不多，但我深感他是一位非常优秀的技术专家和管理者。尤其是在技术方面，他有自己的独特见地，在 IT 软件、硬件及解决方案方面都涉猎很广。另外，他本人也很亲和，具有技术专家的风范。

在最近一次交流中，他跟我提起他打算把之前出版的三本 DB2 系列书籍进行全面版本升级，我感到非常钦佩和欣喜。他在繁忙的日常工作之余，还能利用业余时间完成三本书籍的撰写和更新，足以证明新庄的勤奋和对技术的热爱。同时我也欣喜广大的技术爱好者能有机会一饱技术大家的分享和心得。

他把他的新书送给我，我先粗略读了一遍，更详细的内容留待以后的时间里细细品味。他的这三本书籍将帮助数据库爱好者和企业数据库实践者由浅入深地学习 DB2。即使在网络日益普及的今天，对于一名 DB2 技术工作者来说，通过书籍来系统化地进行学习同样很关键。

在我看来，阅读他的书籍有三个最特别之处：

第一，他是第一位出版 DB2 系列中文书籍的作者，随着这么多年书籍的广泛传播，他在此基础上再次升级更新，结合了非常多的读者反馈，增加了很多近几年读者关注和遇到的问题，这个非常难得。

第二，他所在单位的核心数据库就是 DB2，本次书籍的升级纂写，也更多结合了他的实战经验，这将极大帮助更多企业在应用 DB2 数据库时借鉴和学习。

第三，很多的技术书籍是由专注于技术的工作者纂写，而新庄同时还是非常重要的技术管理和实践者。站在管理者的角度纂写的技术书籍更是融合了管理者如何看待技术的处理和看待问题的视角。

这几本书综合来看，也体现了一个技术管理者乐于分享的心意，这一点是最难得的。

最后，让我表达对新庄的敬意和谢意，感谢他对推动中国的信息化建设和技术的普及所作出的贡献！希望广大的技术爱好者和技术管理者好好品味这些书籍，相信你们一定能从中获益匪浅！

IBM 全球副总裁兼 IBM 软件集团大中华区总经理　胡世忠

序　二

自 1970 年 IBM 公司研究员 E.F.Codd 博士，即“关系数据库之父”，发表业界第一篇关于关系数据库理论的论文 *A Relational Model of Data for Large Shared Data Banks* 以来，伴随着 DB2 的诞生，IBM 公司涌现出了一批优秀的数据库技术领域先驱科学家，并获得了一系列数据库领域大奖，比如在 1981 年荣获了计算机科学界的最高荣誉——ACM 图灵奖。在此之后，数据库管理软件在企业中得到广泛应用，业务流程自动化得以实现，对日常的工作和生活带来了深远的影响。

随着近年来云计算、大数据、移动以及社交信息技术的发展，数据技术也正在经历深刻的变革，处于一个全新计算时代的最前沿。我们能够观察到这样一个趋势：数据库的 24×7 高可用性、高可伸缩性，企业处理海量信息的方式将趋于实时，并从根本上转变业务运作的模式。客户在数据处理速度、简化程度和成本控制等方面需要更上一层楼。最新版的 DB2 V10.1 能带来更低的存储要求以及更高的响应速度，并添加了对大数据管理(如Hadoop)的支持。更具创新性的 PureData 也在这样的技术潮流中应运而生，它整合了基础

架构、统一平台管理和专家知识体系，能够以不同的配置分别提供 OLTP(联机事务处理)、OLAP(联机分析处理)和大数据分析操作的能力。

今天的企业用户希望他们的数据库能够可靠高效地运作，并推动业务发展。当我们把目光放到中国，就会看到，DB2 已成为各行业大型应用系统的支柱产品。但是因 DB2 而闻名业界的本土技术专家，并不多见。

认识新庄是在今年 8 月北京举办的“IBM 软件技术峰会”上。在中国的数据库技术领域，他是许多年轻人的楷模。新庄的成长令人欣喜，新庄对技术和实践的孜孜不倦令人印象深刻。这套 DB2 书籍得益于他历年的钻研及实践，对 DB2 初学者、DB2 管理员以及资深从业者，都有非常好的指导及参考价值。祝愿每一位读者能有所得、有所悟，成长为新一代的数据技术专家，也祝愿新庄在数据技术领域这条康庄大道上走得更宽更远。

IBM 全球副总裁兼 IBM 中国开发中心总经理　王阳

前　　言

DB2 数据库是 IBM 公司关系型数据库核心产品，在国内以及全球有着广泛的应用。针对 DB2 初学者，本书循序渐进地把 DB2 涉及的众多概念和知识介绍给大家。客户端连通性、实例、数据库、表空间和缓冲池、数据移动、备份恢复、SQL 基础知识、DB2 基本监控方法、运行数据库必须考虑的设置、DBA 日常维护以及数据库常用工具都是本书关注的重点。在介绍这些数据库对象和概念的同时，作者尽可能从 DBA 日常工作的角度探究 DB2 数据库常规维护工作。本书同时还就表、索引、序列、触发器等数据库对象从应用设计的角度进行了介绍。本书适合 DB2 的初学者、DB2 开发人员、准备参加 DB2 认证考试的读者以及 DB2 数据库管理人员学习和阅读。

本书结构

本书共 13 章，具体结构如下：

第 1 章：DB2 介绍和安装。在这一章中，除介绍初学者比较熟悉的 Windows 安装外，还花费比较多的篇幅介绍了在 UNIX/Linux 环境下的安装。这主要是因为作者碰到的 DB2 生产环境几乎都是 UNIX/Linux 环境，而在 UNIX/Linux 环境下安装 DB2 时涉及的准备工作又远大于在 Windows 环境下。

第 2 章：创建实例和管理服务器。与其他数据库系统类似，DB2 中也存在实例概念，主要对应 DB2 二进制代码。而管理服务器则是 DB2 中特有的，用于帮助 DBA 对远程主机上的多个实例进行控制。本章详细介绍了实例的创建、删除、配置以及相关的操作系统环境变量等，对于管理服务器，由于在生产实践中使用较少，因而只进行了简单介绍。

第 3 章：创建数据库和表空间。本章介绍了 DB2 数据库的存储模型，创建数据库命令的具体选项对后继工作的影响。本章重点介绍了 DB2 数据库表空间的管理类型，并指出不同类型之间的优缺点。在表空间部分，本章还讲述了影响表空间性能的所有选项，如预取大小、扩展大小等，同时又指出操作系统 IO 设置对表空间性能的影响。与表空间关联的是缓冲池，本章给出了缓冲池的设计与维护原则。

第 4 章：访问数据库。本章介绍了如何配置 DB2 服务器与客户端，使得客户端能够访问服务器上的数据。本章介绍了 DB2 命令行工具 CLP 的使用，同时也讲述了在客户端如何通过各种图形工具配置到服务器的连通性。在这些基础上，本章给出了 DB2 节点目录、数据库目录、本地数据库目录之间的相互关系与区别。

第 5 章：创建数据库对象。本章介绍了常见 DB2 对象的维护方法，重点讲述了数据库中最重要的对象——表的设计考虑。同时本章也介绍了如何使用索引、序列提高性能。

第 6 章：数据移动。在创建表对象后，DBA 的下一步工作就是向表中填充数据。几乎所有系统的构建都涉及数据移动。本章介绍了从数据库中导出数据、向数据库导入数据，重点讲述了 DB2 效率非常高的数据移动工具 LOAD。对于 LOAD 工具，讲述了如何在线 LOAD、如何监视 LOAD、LOAD 性能提高选项、LOAD 异常处理等。在本章中，作者总结了数据移动中经常出现的问题，并给出了相关解决办法。最后，本章介绍了集成数据移动工具 db2move 和数据字典抽取工具 db2look 的使用。

第 7 章：数据库备份与恢复。本章介绍了数据库系统通常碰到的几种备份恢复类型，并指出 DB2 如何配置日志以支持这些类型。本章描述了各种情况下如何重建数据库，同时给出了监控 DB2 数据库备份、恢复进度的方法，以及如何优化备份恢复的速度。

第 8 章：SQL 基础知识。本章针对初学者，介绍了 DB2 中使用的 SQL 语言的基本功能和特点，由于篇幅有限，只介绍了最常用的 DML(Data Manipulation Language)和 DDL(Data Define Language)的语法和示例。最后用较短的篇幅介绍了 DB2 中使用 SQL 的一些最佳实践。

第 9 章：DB2 基本监控方法。DB2 数据库给出了多种手段用于监控数据库的内部运行

情况，如事件监控、快照监控、动态性能视图等。本章主要介绍了实践中使用较多的快照监控，给出了许多生产中的实际案例。

第 10 章：运行数据库必须考虑的数据库设置。在安装、设计完数据库后，并将数据库投入生产环境运行之前，我们必须考虑很多影响数据库运行的设置，否则数据库可能无法运行，可能运行一段时间后将出现各种问题。本章按不同的主题，列出了必须考虑的设置。

第 11 章：DBA 日常运行维护。DBA 的职责是保证数据库稳定、高效运行，除了正常的运行维护外，DBA 还经常碰到各种其他问题，本章主要介绍了作者在日常工作中进行的维护工作。本章首先介绍了如何对 DB2 数据库进行健康性检查以及检查涉及的各个方面，然后给出了找出各种类型的 TOP10 的 SQL 语句方法。

第 12 章：数据库常用工具。本章介绍了 DBA 在日常工作中经常使用的各种工具，如性能解释工具、数据设计建议工具、基准测试工具、数据库一致性检查工具等。熟练掌握这些工具，对 DBA 而言犹如利器在手。

第 13 章：DB2 V10.1 新特性。本章分类介绍 DB2 V10.1 版本提供的各种新功能、新特性，为以后我们开始使用这一新版本提供好的开始。

致谢

本书在出版的过程中得到了清华大学出版社王军编辑的大力支持！这套 DB2 书籍从选题、审稿到出版无不得到他的热心帮助，在此致以深深的谢意！

感谢我的好兄弟骆洪青和袁春光，他们审核了书中的大部分章节，同时也感谢中信银行的胡瑞娟、苏兰芳和我的师弟林春，他们审核了部分章节并从用户的角度给我提出了很多宝贵的建议！

最后，谨以此书献给我慈爱的母亲，母亲从小就教育我努力、正直、踏实和勤奋。正是由于母亲的影响和教育才有了我今天的一点微小的成绩。

目　　录



DB2 介绍和安装

DB2 LUW(IBM DB2 Database for Linux、UNIX and Windows，本后续章节中，本书统一简写为 DB2)是 IBM 于 1983 年推出的第一款面向大型企业的商业化关系数据库管理系统。在 20 世纪 80 年代初，DB2 的发展重点放在大型的主机平台，从 80 年代中期到 90 年代初，DB2 已发展到中型机、小型机以及微机平台。DB2 的诞生不仅促进了与关系数据库概念相关的数学和科学的发展，还创造性地开发出一种极具影响力的全新软件类型。今天，DB2 已经发展成为 IBM 信息管理(IM)软件组合的重要组成部分。在 IBM 信息随需应变策略和体系结构中，DB2 扮演数据基础服务平台的重要角色，并且已经发展成同时支持传统关系数据和 XML 的混合型数据服务器。传承 IBM 数据库的优良传统并具有突破性的数据库产品 DB2 LUW V10.1 已经于 2012 年 4 月份问世，它在原有 V9 版本的基础上，增加了众多革命性的技术，使 DB2 LUW 产品在多个领域实现了突破。

DB2 数据库产品及解决方案广泛应用在金融、电信、制造、零售、保险等行业及政府机关，以数据库技术创新帮助客户实现更大价值，以技术创新推动商业模式的变革和不断发展。

本章主要讲解如下内容：

- DB2 数据库概述
- DB2 数据库的安装和配置
- DB2 数据库的体系结构

1.1 DB2 数据库概述

1.1.1 DB2 发展历史

我们都知道DB2是关系型商用数据库的一种，那么在开始学习DB2之前，先来了解一

下数据库的发展历史。

1. 数据库的发展历史

在没有数据库之前，人们靠什么来记录数据呢？最早是靠文件，但是用文件记录有很多缺点，例如不易保存和共享等，而数据库的出现可以解决这些问题。数据库的历史可以追溯到 40 多年前，当时计算机开始广泛地应用于数据管理，对数据的共享提出了越来越高的要求。传统的文件方式已经不能满足人们的需要。能够统一管理和共享数据的数据库管理系统(DBMS)应运而生。数据模型是数据库系统的核心和基础，各种 DBMS 软件都是基于某种数据模型的。所以通常也按照数据模型的特点将传统数据库系统分成网状数据库、层次数据库和关系数据库 3 类。最早出现的是网状 DBMS，1961 年通用电气公司(General Electric)的 Charles Bachman 成功地开发出世界上第一个网状 DBMS，也是第一个数据库管理系统——集成数据存储(Integrated DataStore，IDS)，奠定了网状数据库的基础，并在当时得到了广泛的发行和应用。IDS 具有数据模式和日志的特征，但它只能在 GE 主机上运行，并且数据库只有一个文件，数据库所有的表必须通过手工编码来生成。网状数据库模型对于层次和非层次结构的事物都能比较自然地模拟，在关系数据库出现之前，网状 DBMS 要比层次 DBMS 用得普遍。在数据库发展史上，网状数据库占有重要地位。层次 DBMS 是紧随网状 DBMS 而出现的。最著名、最典型的层次数据库系统是 IBM 公司在 1968 年开发的 IMS(Information Management System)——一种适合其主机的层次数据库。这是 IBM 公司研制的最早的大型数据库系统产品。从 20 世纪 60 年代末产生起，如今已经发展到 IMS V10，提供对群集、N 路数据共享、消息队列共享等先进特性的支持。这个具有 40 年历史的数据库产品在如今的 WWW 应用连接、商务智能应用中仍扮演着新的角色。目前国内的 4 大银行在主机上仍然在使用 IMS 数据库。

关系数据库

关系数据库的由来：网状数据库和层次数据库已经很好地解决了数据的集中和共享问题，但是在数据独立性和抽象级别上仍有很大欠缺。用户在对这两种数据库进行存取时，仍然需要明确数据的存储结构，指出存取路径，而后来出现的关系数据库较好地解决了这些问题。1970 年，IBM 的研究员 E.F.Codd 博士在刊物 *Communication of the ACM* 上发表了一篇名为“A Relational Model of Data for Large Shared Data Banks”的论文，提出了关系模型的概念，奠定了关系模型的理论基础。尽管之前在 1968 年 Childs 已经提出了面向集合的模型，然而这篇论文被普遍认为是数据库系统历史上具有划时代意义的里程碑。Codd 的心愿是为数据库建立优美的数据模型。后来 Codd 又陆续发表多篇文章，论述了范式理论和衡量关系系统的 12 条标准，用数学理论奠定了关系数据库的基础。IBM 的 Ray Boyce 和 Don Chamberlin 将 Codd 关系数据库的 12 条准则的数学定义以简单的关键字语法表现出

来，里程碑式地提出了 SQL(Structured Query Language)语言。关系模型有严格的数学基础，抽象级别比较高，而且简单清晰，便于理解和使用。但是当时也有人认为关系模型是理想化的数据模型，用来实现 DBMS 是不现实的，尤其担心关系数据库的性能难以接受，更有人视其为当时正在进行中的网状数据库规范化工作的严重威胁。为了促进对问题的理解，1974 年 ACM 牵头组织了一次研讨会，会上开展了一场分别以 Codd 和 Bachman 为首的支持和反对关系数据库两派之间的辩论。这次著名的辩论推动了关系数据库的发展，使其最终成为现代数据库产品的主流。而 Oracle 公司在 1979 年开发了第一个商用 SQL 关系数据库管理系统。关系数据库系统以关系代数为坚实的理论基础，经过几十年的发展和实际应用，技术越来越成熟和完善，代表产品有 Oracle、DB2、SQL Server 以及 Informix、Sybase 等等。

面向对象数据库

随着信息技术和市场的发展，人们发现关系数据库系统虽然技术很成熟，但其局限性也是显而易见的——它能很好地处理所谓的“表格型数据”，却对技术界出现的越来越多的复杂类型的数据无能为力。20世纪90年代以后，技术界一直在研究和寻求新型数据库系统。但在什么是新型数据库系统的发展方向的问题上，产业界一度是相当困惑的。受当时技术风潮的影响，在相当一段时间内，人们把大量的精力花在研究“面向对象的数据库系统(Object-Oriented Database)”或简称“OO数据库系统”。值得一提的是，由美国Stonebraker教授提出的面向对象的关系数据库理论曾一度受到产业界的青睐。而Stonebraker本人也在当时被Informix花大价钱聘为技术总负责人。然而，数年的发展表明，面向对象的关系数据库系统产品的市场发展情况并不理想。理论上的完美性并没有带来市场的热烈反应。其不成功的主要原因在于，这种数据库产品的主要设计思想是企图用新型数据库系统来取代现有的数据库系统。这对许多已经运用数据库系统多年并积累了大量工作数据的客户，尤其是大客户来说，是无法承受因为新旧数据间的转换而带来的巨大工作量及巨额开支的。另外，面向对象的关系数据库系统使查询语言变得极其复杂，从而使得无论是数据库的开发商还是应用客户，都视其复杂的应用技术为畏途。

混合数据库

IBM 经过多年的积累和持续创新，在 2006 年年底率先推出了第一个直接支持 XML 的混合数据服务器——IBM DB2 V9(代号为 Viper)。

IBM DB2 V9 提供了与以前版本非常不同的体系结构，它通过提供新的查询语言、新的存储技术、新的索引技术和支持 XML 数据及其固有层次结构的特性，使得 IBM DB2 V9 成为 IBM 的第一个“混合型”(即多结构)数据库管理系统。除了支持表数据模型外，DB2 还支持 XML 文档和消息中固有的层次化数据模型。用户可以在表中自由地混合存储传统

SQL 数据和最新的 XML 数据。还可以使用 SQL(如果愿意，可以加上 XML 扩展)和 XQuery(新出现的 XML 数据查询标准)来查询和处理这两种形式的数据。在经过实践检验的数据库管理基础设施上进行扩展，IBM 为 DB2 V9 用户提供了同时处理关系数据和 XML 数据的强大支持。新的基于 XML 的应用程序使用 DB2 可以无缝地访问关系数据库资源，拥有企业级 XML 应用的访问能力。DB2 在所有特性/接口中都整合了对 XML 数据的支持，而 DB2 对 XML 的“固有”支持是在对其他技术的现有支持之外提供的，SQL、表格数据结构和各种 DBMS 现有特性仍然能够获得最好的支持。因此，用户可以用一个数据库对象同时管理“传统的”SQL 数据和 XML 文档。

为了高效地管理传统 SQL 数据类型和 XML 数据，DB2 包含两种不同的存储机制。但是，一定要注意，给定数据类型所用的底层存储机制对于应用程序是透明的。换句话说，应用程序不需要显式地指定要使用的存储机制，也不需要管理存储的物理细节，比如如何将 XML 文档的各个部分拆分到多个数据库页上。系统会自动采用适合目标数据的格式，应用程序自然而然地获得存储和查询数据方面的运行时性能优势。DB2 V9 的存储模型如图 1-1 所示。

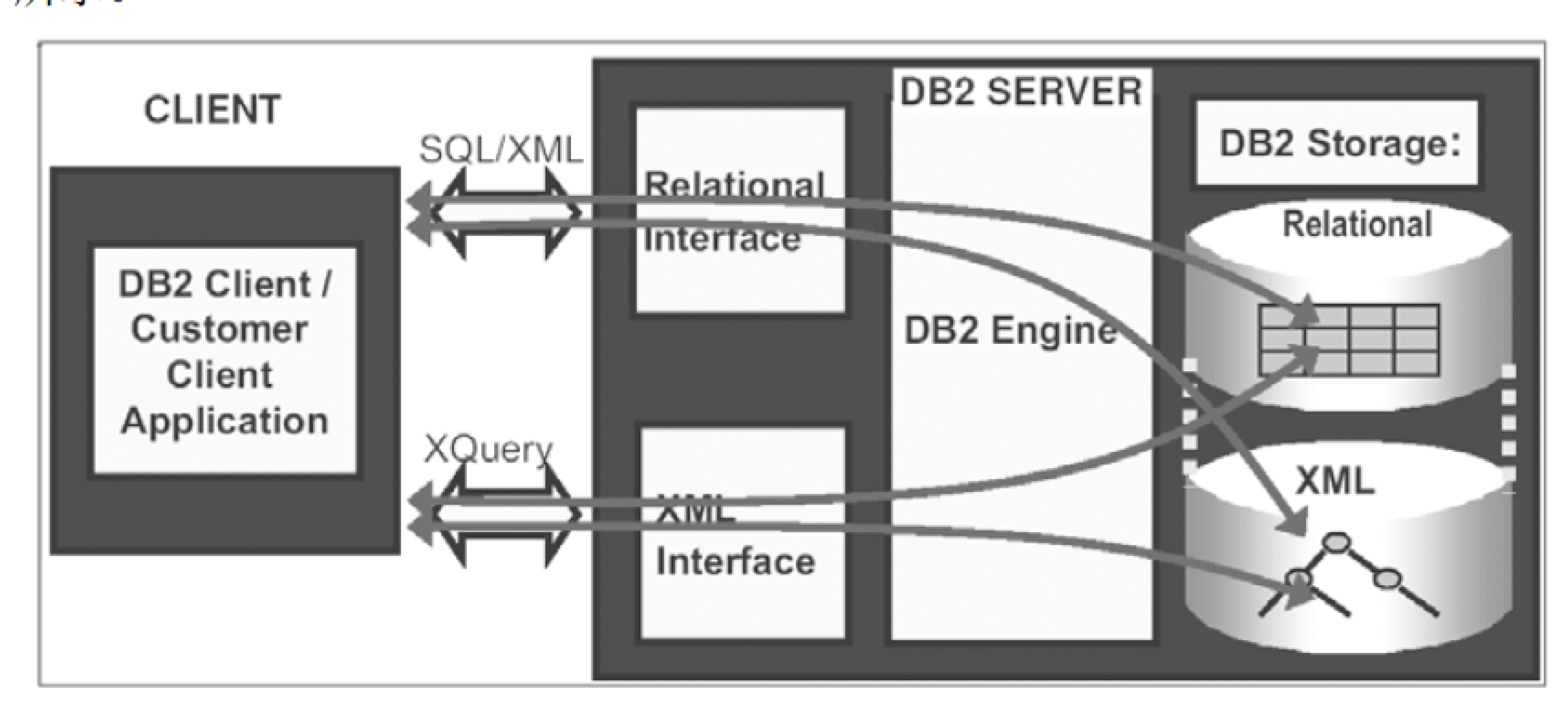


图 1-1 DB2 V9 的存储模型

2. 数据库大事年鉴

- 1969 年：IBM 开发的层次数据库 IMS 诞生。这是 IBM 公司的第一代数据库，也叫 DB1。
- 1970 年：IBM 的 E.F.Codd 发表了关于关系数据库技术的第一篇论文“基于大型共享数据库的数据关系模型”。
- 1973 年：在 IBM 研究中心确立了 System R 项目，建立关系数据库管理系统。

- 1974 年：IBM 的 Don Chamberlin 和 Ray Boyce 发表了“EQUEL：结构化英语查询语言”，成为 SQL 标准定义的基础。
- 1975 年：IBM 的 Don Chamberlin 和 Morton Astrahan 发表的论文——“结构化英语查询语言的实现”阐述了作为 System R 一部分的第一个 SQL 实现。
- 1976 年：IBM System R 团队发表了“System R：数据库管理的关系方法”，这篇论文阐述了他们的关系数据库原型。IBM 的 Jim Gray 发表的“共享数据库中的锁粒度和结合度”，阐述了数据库事务和结合度的正式定义，这是数据库并发理论的基础。
- 1979 年：IBM 的 Pat Selinger 在她的论文“关系数据库管理系统中的访问路径选择”中阐述了行业的第一个关系查询优化器，这是 DB2 数据库优化器的雏形。
- 1979 年：Oracle 公司开发了第一个商用 SQL 关系数据库管理系统。
- 1980 年：在一家早期的 S-100/CP/M 公司 Cromemco 工作的 Roger Sippl 和 Laura King 开发了一个基于 ISAM 技术的小型关系数据库，作为一款报表记录器软件的一部分。1980 年，Sippl 和 King 离开 Cromemco 去开发关系数据库系统(RDS)。1981 年发布了他们自己的产品——Informix(INFORMation on unIX)。
- 1982 年：IBM 为 VSE/VM 发布 SQL/DS，作为第一个商业用途的关系数据库(带有基于 System R 的 SQL 接口)。SQL/DS 是 DB2 数据库的前身。
- 1984 年：Sybase 公司成立，公司名称“Sybase”取自“system”和“database”相结合的含义。Sybase 公司的创始人之一 Bob Epstein 是 Ingres 大学版(与 System/R 同时期的关系数据库模型产品)的主要设计人员。Sybase 公司的第一个关系数据库产品是 1987 年 5 月推出的 Sybase SQLServer 1.0。Sybase 首先提出 Client/Server 数据库体系结构的思想，并率先在 Sybase SQLServer 中实现。
- 1988：SQL Server 由微软与 Sybase 共同开发，最早运行于 OS/2 平台。所以您会发现 SQL Server 和 Sybase 数据库早期的架构基本上一样。直到 SQL Server 2005 以后，微软才做了大幅度改动。

1.1.2　DB2 版本和平台支持

1. DB2 平台支持

如图 1-2 所示，DB2 产品几乎覆盖了当前所有流行的硬件和操作系统平台。在大型机操作系统上，DB2 for z/OS (DB2 for OS/390、DB2 for MVS/ESA、DB2 for VM/VSE)利用了 System z 服务器上的硬件耦合器(Coupling Facility)，因此与使用“shared-nothing”方式的 DB2 LUW 相反，它采用“shared-everything”方式。

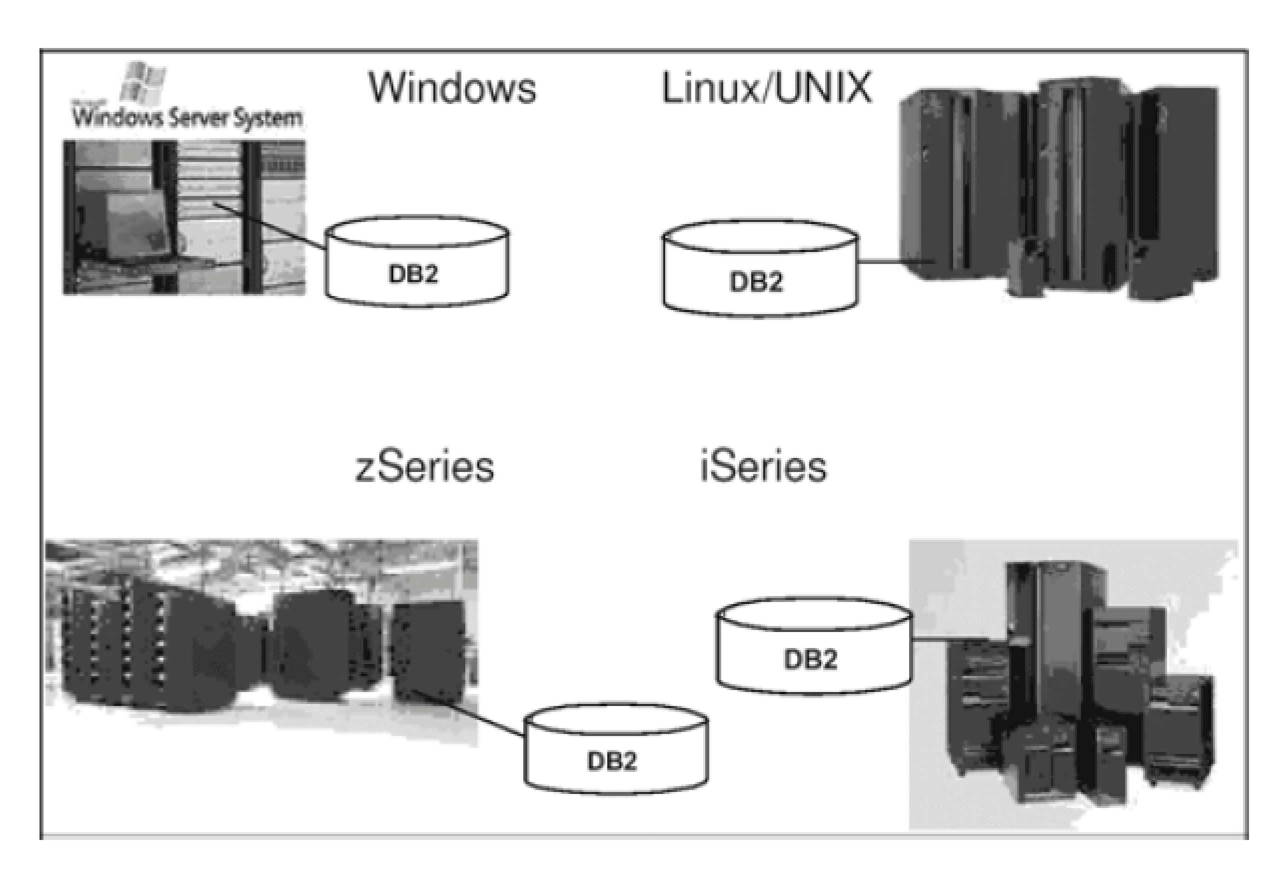


图 1-2　DB2 支持的平台

在由 IBM 公司设计的中型机上(AS/400)，有 DB2 for System i，DB2 已经嵌入在 i5/OS 操作系统中，成为其不可分割的一部分。DB2 对 UNIX 操作系统的支持同样十分广泛，可以在 AIX、HP-UX、Solaris 等多种系统上找到相应的版本。

在 PC 操作系统上，DB2 可以对 Windows 9x、Windows NT、Windows XP 以及 Windows Vista 等多种操作系统提供支持。同样，DB2 从版本 6.1 后也增加了对 Linux 的支持。

以上我们提到的只是 DB2 服务器所能运行的平台，DB2 的客户端所能支持的平台更为广泛，除了以上提到的所有平台之外，DB2 的客户端还能运行在 DOS、Windows 3.x、Mac OS 以及 SGI 公司的 IRIS 系统之上。综上所述，DB2 数据库服务器对平台的支持主要有 3 种：DB2 for z/OS 大型机平台、DB2 for i5/OS 中型机平台和 DB2 for LUW (DB2 for Linux、DB2 for UNIX 和 DB2 for Windows)开放平台。最早的 DB2 产品是 DB2 for MVS/ESA，以后的产品设计都延续了它的基本结构及关键算法，保证了不同系统之间的可移植性、可扩展性和互操作性。但是，由于不同操作系统之间存在着不小的差异，DB2 系列产品还针对相应的平台进行了一定优化，以适应各个操作系统的特性。在本书我们主要讲解 DB2 for LUW 平台。

注意：
以后在本书中我们要提及的 DB2，如无特别说明指的是 DB2 for LUW 平台。

2. 版本支持

DB2 产品除了能够对各种硬件和操作系统平台进行支持之外，DB2 V9 提供了适于所有企业的数据管理解决方案。为了适应不同规模企业和用户的需要，DB2 提供了不同级别

的产品，对小到个人用户、大到跨国企业的不同需求提供支持。没有其他数据库管理系统能够在性能、可用性、可伸缩性和可管理性方面达到 DB2 V9 的水平。

DB2 有不同的版本，每种版本适合不同需求的用户。图 1-3 显示了所有可用的 DB2 分发版本。从图中可以看出，DB2 的每个高版本都包含低一级版本的所有功能和特性，并添加了新的特性和功能。Linux、UNIX 和 Windows(LUW 平台)上的代码有大约 90%是相同的，在每种操作系统上有 10%的专用代码，用于使数据库与底层操作系统紧密地集成。例如，使用 AIX 上的 JFS2 文件系统或 Windows 上的 NTFS 文件系统。

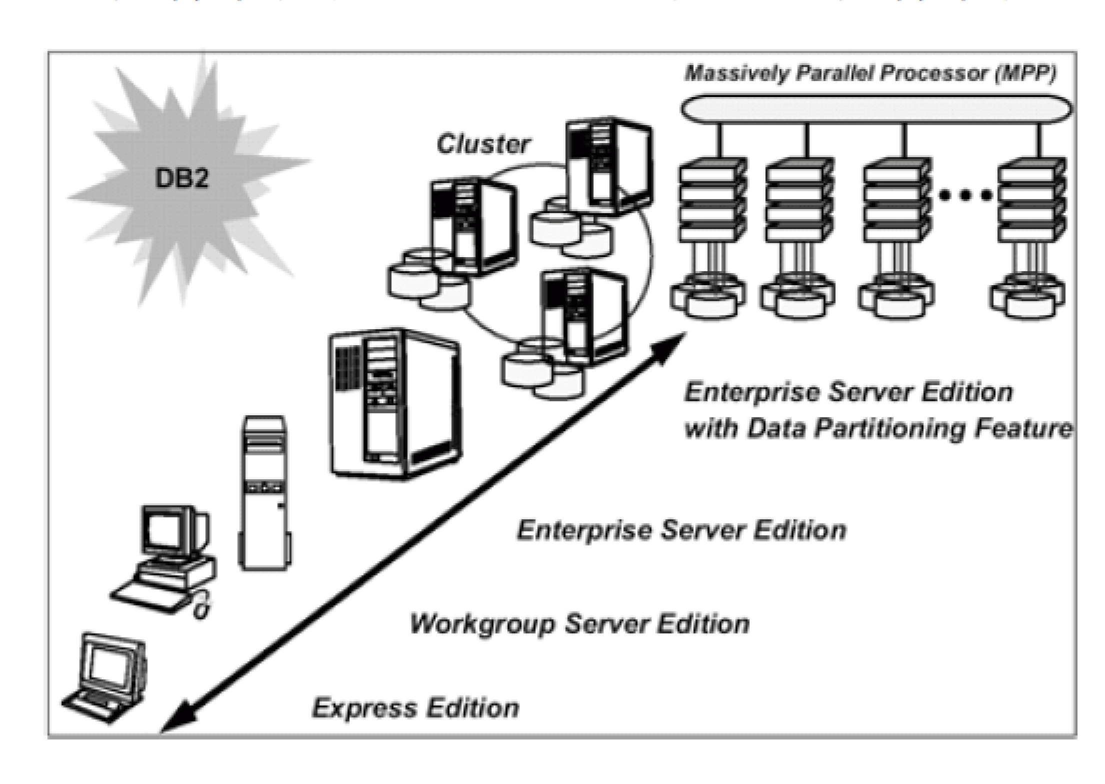


图 1-3　DB2 版本分类

以下是对 DB2 V9 版本中不同级别产品的特点介绍：

DB2 V9 为各种类型的业务提供了正确的数据管理解决方案。提供封装了众多特性和功能的不同产品版本，以适应大量来自客户的不同需求。小型企业可以选择 Express Edition，中型企业可以选择 Workgroup，而 Enterprise Edition 则适合大型企业。除了这些版本，DB2 V9 另外提供了两个版本：Personal Edition 和面向开发的 Developer Edition，以及免费版 DB2 Express-C。表 1-1 描述了 DB2 V9 可用的版本。

表 1-1　DB2 V9 版本和平台支持

DB2 V9 分发版本
DB2 Express Edition V9 for Linux、UNIX and Windows DB2 Express V9 是功能完备的 DB2 数据服务器，它为中小企业(Small and Medium Business，SMB)市场提供了极具吸引力的入门级价格。该版本提供了经简化的程序包，可在应用程序内轻松进行透明安装。DB2 Express V9 可以轻松升级到 DB2 V9 的其他版本，它还具有和其他可伸缩性更高的版本相同的自主管理特性。DB2 Express V9 合并了本地 XML 数据存储，并允许使用 XQuery、XPath、SQL 和标准报告生成工具来灵活地访问 XML 数据

(续表)

DB2 V9 分发版本
DB2 Workgroup Server Edition V9 for Linux、UNIX and Windows DB2 工作组服务器版用来满足数据服务器部署工作组或中型业务环境的需要。DB2 工作组服务器版包含了本地 XML 数据存储，并允许使用 XQuery、XPath、SQL 和标准报告生成工具来灵活地访问 XML 数据
DB2 Enterprise Server Edition (ESE) V9 for Linux、UNIX and Windows DB2 企业服务器版用来满足数据服务器处理大中型业务的需要。可以将它部署在任意大小(从一个 CPU 到任意数目的 CPU)的 Linux、UNIX 或 Windows 服务器上。DB2 企业服务器版是用于构建随需应变的企业级解决方案的理想平台，这些解决方案的示例如下：大小为 TB 级的大型数据仓库，高性能(每周 7 天，每天 24 小时全天候运行)的高容量事务处理业务解决方案，或是基于 Web 的解决方案。DB2 企业服务器版包含本地 XML 数据，并允许使用 XQuery、XPath、SQL 和标准报告生成工具来灵活地访问 XML 数据。DB2 企业服务器版具有可选功能部件，用来在诸如数据库分区、性能、安全性、数据联合以及数据库管理方面提供附加的高级产品功能。此外，DB2 ESE V9 还提供了与其他 Enterprise DB2 和 Informix 数据源的连通性、兼容性以及集成
DB2 Enterprise Server Edition (ESE) V9 for Linux、UNIX and Windows With DPF DB2 ESE V9 With DPF 可以构建分区数据库，可以构建基于 MPP 的集群结构，主要应用于高性能计算领域，例如数据仓库、科学计算、人工智能等
DB2 Personal Edition for Linux、UNIX and Windows DB2 Personal V9 是单用户、功能完备、具有内置复制的关系数据库。对于基于桌面和膝上型电脑的部署是理想选择。DB2 Personal V9 可以进行远程管理，这使其成为在不要求多用户能力的不定期连接或远程办公实现中的最佳部署选择
Database Enterprise Developer Edition 此版本为单一应用程序开发人员提供软件包，用于设计、构建和原型化应用程序，以在任意 IBM 信息管理客户端或服务器平台上部署。可以面向 DB2 所有平台的开发。用于 DB2 的一组应用程序驱动程序包括嵌入式 SQL、ODBC/CLI、JDBC/SQLJ、OLEDB、.NET、PHP、Perl 和 Ruby。数据访问和管理工具提供了 DB2 控制中心(Windows 和 Linux)和 DB2 命令行处理器(CLP)
DB2 Express-C DB2 Express-C 是为社区提供的 DB2 Express Edition(DB2 Express)的版本之一。DB2 Express-C 是免费的数据服务器，可用于开发和部署 XML、C/C++、Java、.NET 和 PHP 应用程序。DB2 Express-C 最多可运行在双核 CPU、4GB 内存的服务器上，以及对数据库规模或其他人为限制没有要求的任何存储系统

关于这些版本的详细许可协议超出了本书讨论的范围，但是需要注意每个版本的功能特性是不一样的。某些实用程序和功能仅在特定 DB2 数据库产品版本中可用。在某些情况下，实用程序或功能与特定 DB2 功能部件相关联，如果 DB2 Express 或 DB2 Workgroup 中

没有免费包含某一功能，那么(在大多数情况下)可以通过附加的功能部件(Feature Pack)购买这一功能。

注意：

DB2 的分发版本就像我们使用 Windows XP 操作系统一样，有 Windows Home Edition、Windows XP Professional Edition 和 Windows XP 64-Bit Edition 等版本，每个版本包含的功能不一样，用户可根据自己的需求来决定使用什么版本。本书的内容主要针对 DB2 V9.7 版本，对于其他版本，如有不同之处，本书会给予特别说明。本书所附示例也都经过 DB2 V9.7 版本的验证。

1.1.3　DB2 产品组件和功能

DB2 数据库的产品组件如图 1-4 所示。

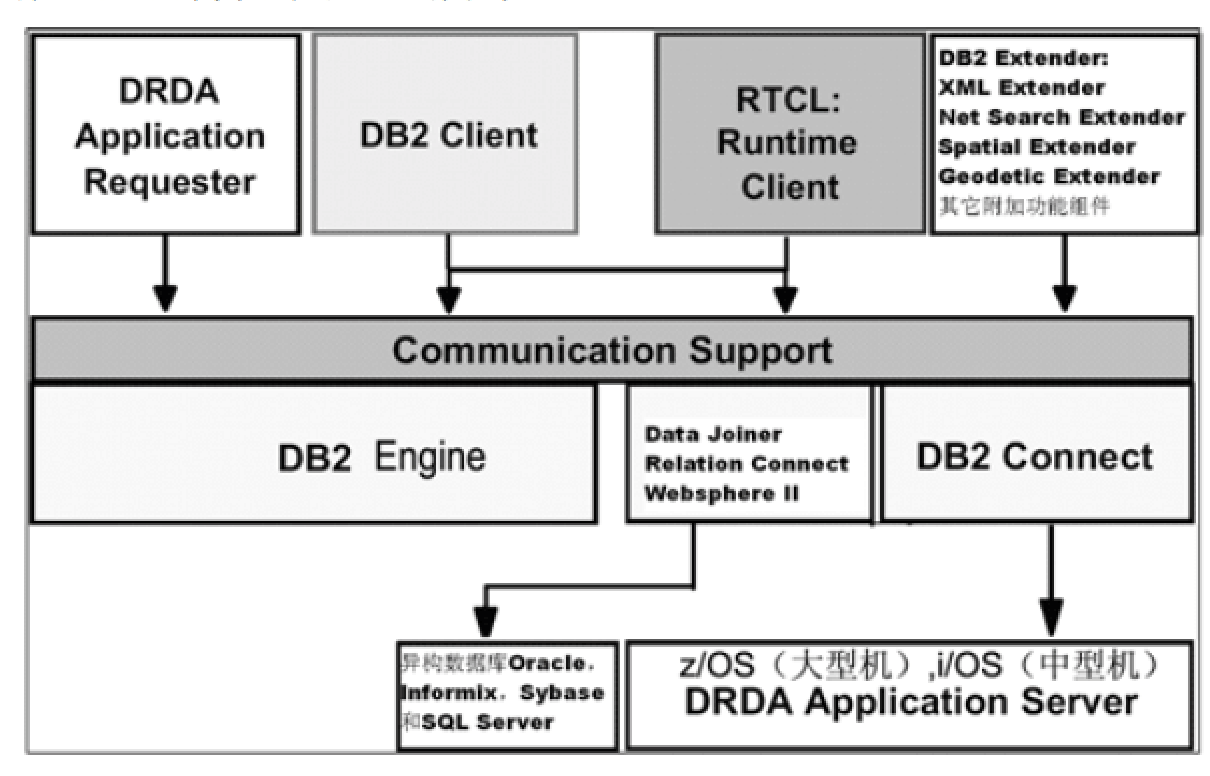


图 1-4　DB2 产品组件

DB2 Engine

DB2 Engine 是整个数据库系统的核心，提供了 DB2 的基本功能。DB2 引擎类似汽车的发动机，负责管理和控制对数据的存取；负责生成程序包(存储存取计划的数据库对象)；提供事务的管理；保障数据的完整性和数据保护；提供应用程序并发控制。数据库引擎(DB2 Engine)设计的完善与否，决定了数据库系统是否稳定和高效。DB2 Engine 是所有数据库中最强大的数据库引擎。

DB2 客户机

DB2 V9 大大简化了将应用程序连接到 DB2 数据库所需的基础设施的部署。DB2 V9 提供以下客户机：

- DB2 V9 Runtime Client (DB2 RTCL)

如果只需要让应用程序能够访问 DB2 V9 数据服务器，那么这就是最佳选择。它们提供了执行此任务所需的 API，但是这种客户机没有提供管理工具。DB2 运行时客户机(DB2 Runtime Client)的前身为 CAE(Client Application Enabler)，是所有 DB2 产品所共用的部件，它允许远程应用程序对数据库服务器进行存取。在这个组件中包含了 CLP(Command Line Processor)，允许用户动态地执行 SQL 语句和 DB2 命令，对本地和远程的数据库服务器进行存取。另外，Runtime Client 还提供了对 ODBC 和 JDBC 的支持，允许用户开发的 ODBC 或 JDBC 应用程序对数据库进行存取。要想对数据库服务器进行存取，Runtime Client 几乎是必不可少的(在 WWW 上通过 Java Applet 存取是唯一的例外)。

DB2 运行时客户机为运行各种平台的工作站提供了访问 DB2 数据库的能力，它只提供基本连通性——不多不少刚刚好。如果要建立到远程 DB2 服务器或 DB2 连接网关(Connect Gateway，可帮助访问类似 DB2 for z/OS 的大型机或主机系统上的 DB2)的连通性，那么至少必须使用这个客户机。当然，也可以使用任何客户机进行连接。Runtime Client 的安装取决于操作系统。比如，如果需要在 Windows 操作系统上对数据库管理器进行存取，就需要在 Windows 系统上安装 Runtime Client for Windows。

- DB2 V9 Client

包含 DB2 Runtime Client 中的所有功能，还增加了通过一组图形化工具进行客户机-服务器配置、数据库管理和应用程序开发的功能。DB2 V9 Client 整合了 DB2 V8 Application Development 和 DB2 V8 Administration Client 中的功能(DB2 管理客户机(Administration Client)是客户端的管理工具，包含了一系列的图形化工具，用户可以方便地通过这些工具对数据库服务器进行远程管理。关于这些图形化的管理工具，我们在第 4 章会有比较详细的介绍)。Administration Client 的安装也取决于操作系统。这类客户机为各种平台的工作站提供了通过控制中心或配置助手(Configuration Assistant)访问并管理 DB2 数据库的能力。DB2 管理客户机具有 DB2 运行时客户机的所有功能，并且还包含所有 DB2 管理工具、文档以及对瘦客户机的支持。DB2 应用程序开发客户机(DB2 Application Development Client)是专门为应用程序开发人员提供的，它包含了开发数据库应用程序所需要的各种组件，包括预编译器、Runtime Client、include 文件、库函数、样例程序和帮助文档等。

- Java Common Client (JCC)

这是 2MB 的可重新发布的客户机，提供了对 DB2 数据服务器的 JDBC 和 SQLJ 应用程序访问，而不需要安装和维护 DB2 客户机代码。如果要连接 DB2 for System i 或 DB2 for System z 数据服务器，那么仍然需要安装 DB2 Connect 产品。

- DB2 V9 Client Lite

这个客户机是 DB2 V9 中新增的，可以执行与 JCC 客户机相似的功能，但是不支持对

DB2 数据服务器进行基于 Java 的访问，而是用于 CLI/ODBC 应用程序。这个客户机尤其适合于那些希望将连接功能嵌入应用程序，而不需要重新发布和维护 DB2 客户机代码的 ISV。

通信支持(Communication Support)

通信支持提供了远程客户机支持，客户端应用程序可以通过多种网络协议对数据库服务器进行存取，TCP/IP、SNA(APPC/APPN)、NETBIOS 和 Name Pipe 等协议都可以被 DB2 所支持。

DB2 Relational Connect，DB2 Data Joiner

DB2 Relational Connect 和 DB2 Data Joiner 通过允许用户和应用程序存取存储在 Oracle、Sybase、Informix、SQL Server 等数据库中的数据，增强了 DB2 数据库中所包括的分布式请求功能。它们允许将驻留在多个不同平台上的数据作为单个数据库映像访问，这些数据既可以是 IBM 的，也可以是其他供应商的；既可以是关系型的，也可以是非关系型的。这可以与内置分布式查询功能一起使用，从而使 DB2、Oracle、Sybase、Informix、SQL Server 等数据库之间的查询 SQL 化。这些组件可以帮助企业整合数据，从而加速 Oracle、Sybase、Informix、SQL Server 等数据库源中的选择性能，以便进行数据集中。

这两个功能今天已经被整合成 IBM 的一个产品——WebSphere Information Integrator。

DB2 Connect

许多大型组织中的大量数据由 DB2 for i5/OS、DB2 for z/OS 等数据服务器进行管理。有了 DB2 Connect 的帮助，在任何支持的 DB2 分布式平台上运行的应用程序都可以透明地操作这些数据，就像是本地数据服务器在管理数据一样。还可以将 DB2 Connect 及其相关工具与许多现成的或定制开发的数据库应用程序一起使用。DB2 Connect 提供了从 Windows、Linux 和 UNIX 开放平台连接大型机和中型机的能力。

DB2 扩展器(DB2 Extender)

DB2 扩展器使数据库应用程序能够超越传统的数字和字符数据，为底层数据服务器提供额外的功能。这部分组件是可选的，包括：DB2 XML Extender 扩展器(处理 XML)；DB2 Net Search Extender 扩展器可以帮助企业在搜索数据库中的数据时获得更高的性能；DB2 Spatial Extender 扩展器可以在 DB2 中与文本和数字等传统数据一起存储、管理和分析空间数据——关于地理特征位置的信息；DB2 Geodetic Extender 扩展器可以将地面作为球体对待，从而消除投影等操作造成的不精确。

注意：

大家了解这些可选的扩展器即可，通常情况下用不到这些组件。

通过上面的讲解，我们大概了解了 DB2 数据库的产品组件和功能，其实有些组件我们很少用到，就像 DB2 Connect，如果单位里没有大型机或中型机，那么可能不会用到。

另外，IBM 在已经发布的 DB2 V10.1 版本中增加了一些组件和功能，我们会在本书的第 16 章介绍相关内容。如果关心相关内容，可以直接阅读第 16 章。

1.2　DB2 数据库的安装与配置

在了解完 DB2 的相关版本和产品组件后，下面开始我们的 DB2 学习之旅。在学习之前，我们要首先安装 DB2 数据库，下面讲解如何在不同的平台上安装 DB2，这部分内容相对简单，为了节省篇幅，我们把部分图形化界面省去了。

DB2 安装方法

表 1-2 列出了不同操作系统上 DB2 的安装方法。

表 1-2　不同操作系统可用的安装方法

安 装 方 法	Windows	Linux 或 UNIX
“DB2 安装”向导(图形化界面)	是(setup.exe)	是(db2setup)
响应文件安装(静默安装)	是	是
db2_install 命令(命令行界面)	否	是(db2_install)

“DB2 安装”向导(db2setup)

“DB2 安装”向导是可在 Linux、UNIX 和 Windows 操作系统上使用的 GUI 安装程序。“DB2 安装”向导提供了易于使用的界面，用于安装 DB2 产品和执行初始设置与配置任务。Linux/UNIX 上启动时需要配置 JRE 运行环境和 X 环境。

“DB2 安装”向导还可以用来创建 DB2 实例和响应文件，它们可用于在其他机器上复制此安装。这种安装方法比较简单，一般用得最多。

响应文件安装

响应文件是包含设置和配置值的文本文件。DB2 安装程序将读取这些文件，并根据已指定的值来执行安装。响应文件安装也称为静默安装。响应文件的另一个优点是：它们提供了对那些不能使用“DB2 安装”向导设置的参数的访问。在 Linux 和 UNIX 操作系统上，如果将 DB2 安装映像嵌入你自己的应用程序中，那么你的应用程序有可能从安装程序中以

计算机可读的格式接收安装进度信息和提示。这种安装方式一般比较适合大批量的客户端安装。

db2_install 命令(仅适用于 Linux 和 UNIX 平台)

db2_install 命令将安装指定的具有“字符”界面支持的 DB2 产品的所有组件。通过使用-L 参数就可以选择要支持的其他语言。尽管 db2_install 命令会安装指定的 DB2 产品的所有组件，但却不会执行用户和组创建、实例创建或配置。在安装之后执行配置时，此安装方法可能是首选。如果希望在安装 DB2 产品时就加以配置，那么请考虑使用“DB2 安装”向导。

1.2.1　DB2 在 Windows 上的安装

在 Windows 上安装 DB2 非常简单，安装步骤如下：

首先检查系统的硬件资源是否满足安装的最小需求，然后启动DB2 安装向导(Windows)。

(1) 用管理员账号登录 Windows 系统。

(2) 关闭所有应用程序，以便 DB2 安装过程可以快速完成并且在安装过程出现问题时易于定位。

(3) 把 DB2 的安装光盘插入光驱，启动自动安装向导，如图 1-5 所示。

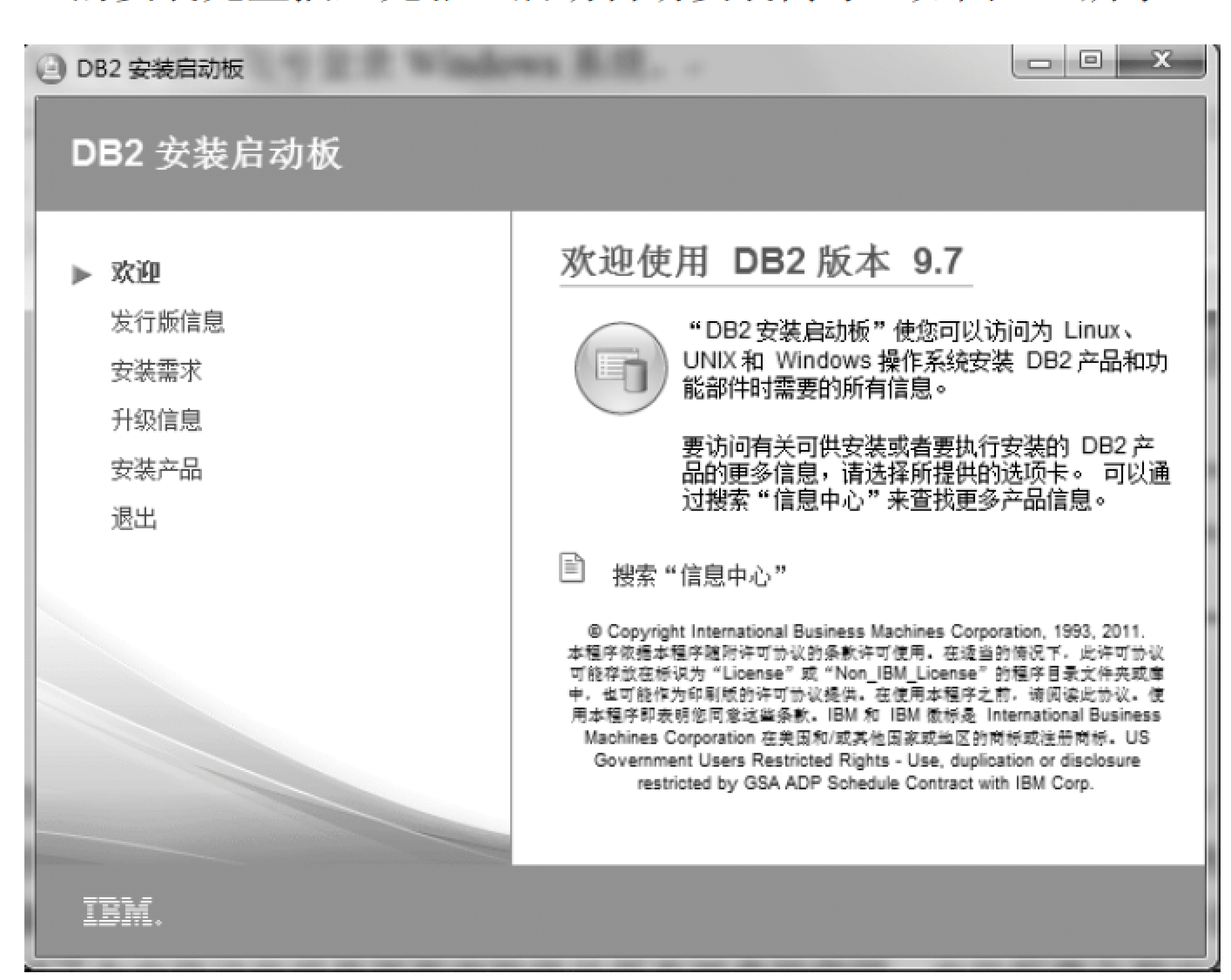


图 1-5　启动“DB2 安装启动板”

在 DB2 安装向导的启动界面中，可以看到左边菜单有“发行版信息”、“安装需求”、“升级信息”、“安装产品”和“退出”这几个选项。可以预先浏览“安装需求”，以了解安装 DB2 有哪些具体要求和注意事项。这里需要强调的是，如果安装选择的组件所需的空间超出为安装这些组件而指定的路径所在磁盘的空间，安装程序就会发出关于空间不足的警告。可以继续安装。但是，如果实际上没有足够的空间用于正要安装的文件，当没有更多的空间时，安装将停止。此时，如果不能释放空间，就必须人工停止安装程序。

浏览“发行版信息”，就可以了解到与 DB2 有关的信息指南。

(4) 单击“安装产品”。

(5) 选择需要安装的版本(本例中我们安装 ESE 版本)，如图 1-6 所示。单击“下一步”按钮继续安装。

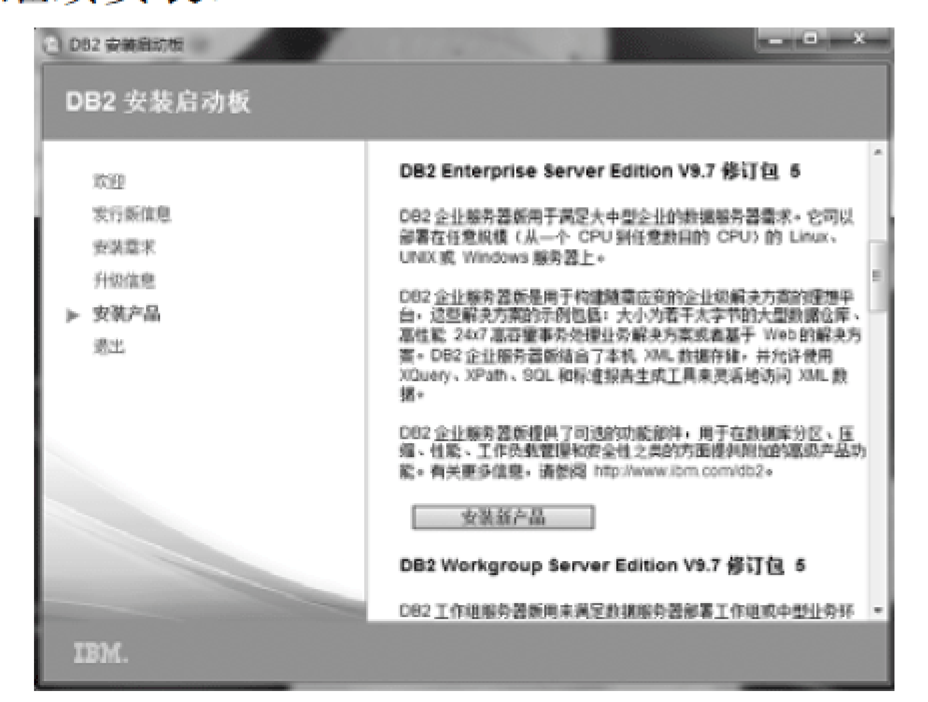




图 1-6　DB2 安装

(6) 阅读并接受许可协议(选中“我接受...”单选按钮)，如图 1-7 所示。单击“下一步”按钮继续。



图 1-7　阅读并接受许可协议

(7) 选择安装类型。

对于本书，选择“典型安装”选项(这是默认设置)，如图 1-8 所示。“精简安装”选项执行基本安装，而“定制安装”选项允许用户定制希望安装的特性。单击“下一步”按钮继续。

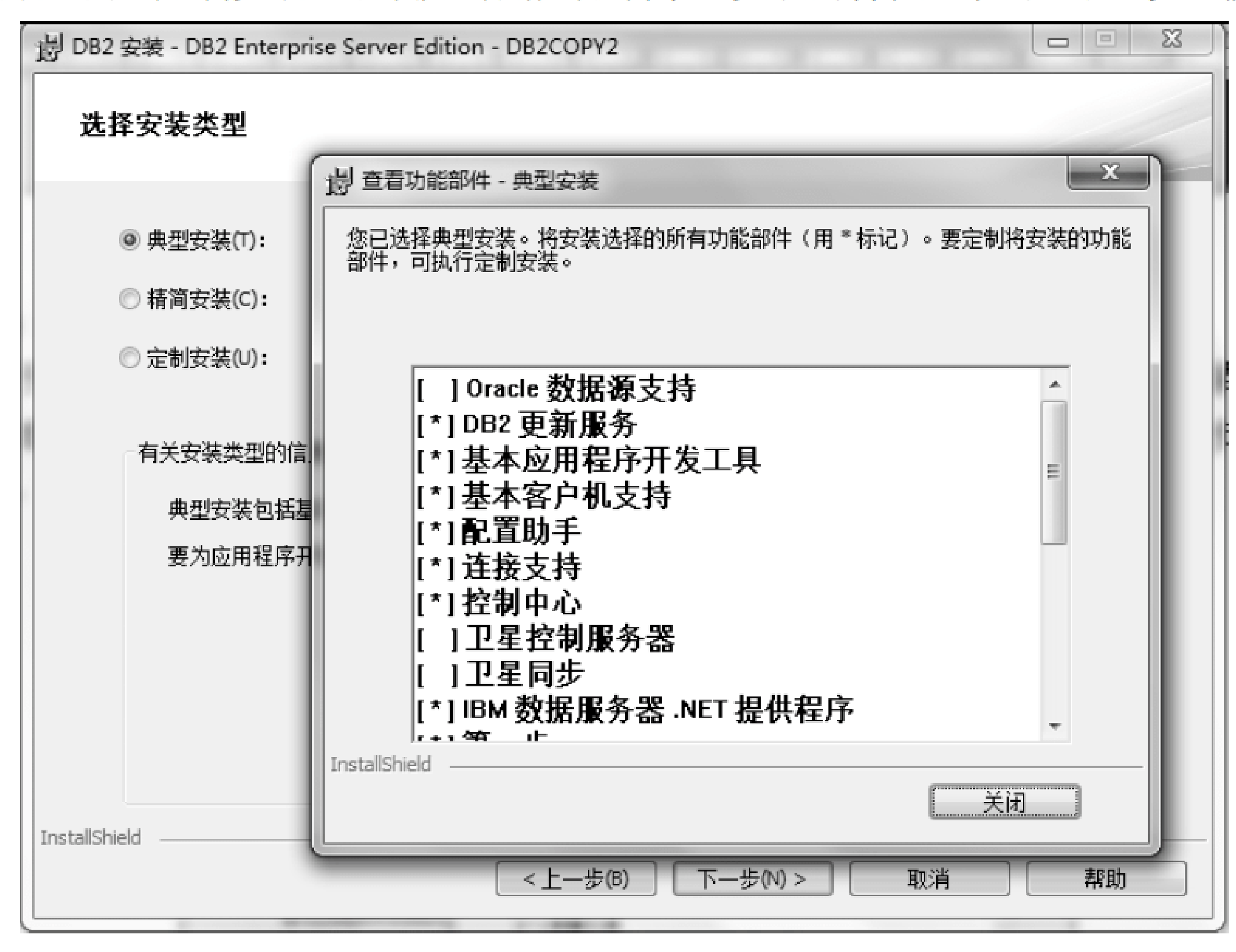


图 1-8　选择安装类型

可以选择是否创建响应文件以便日后执行响应文件安装，如图 1-9 所示。

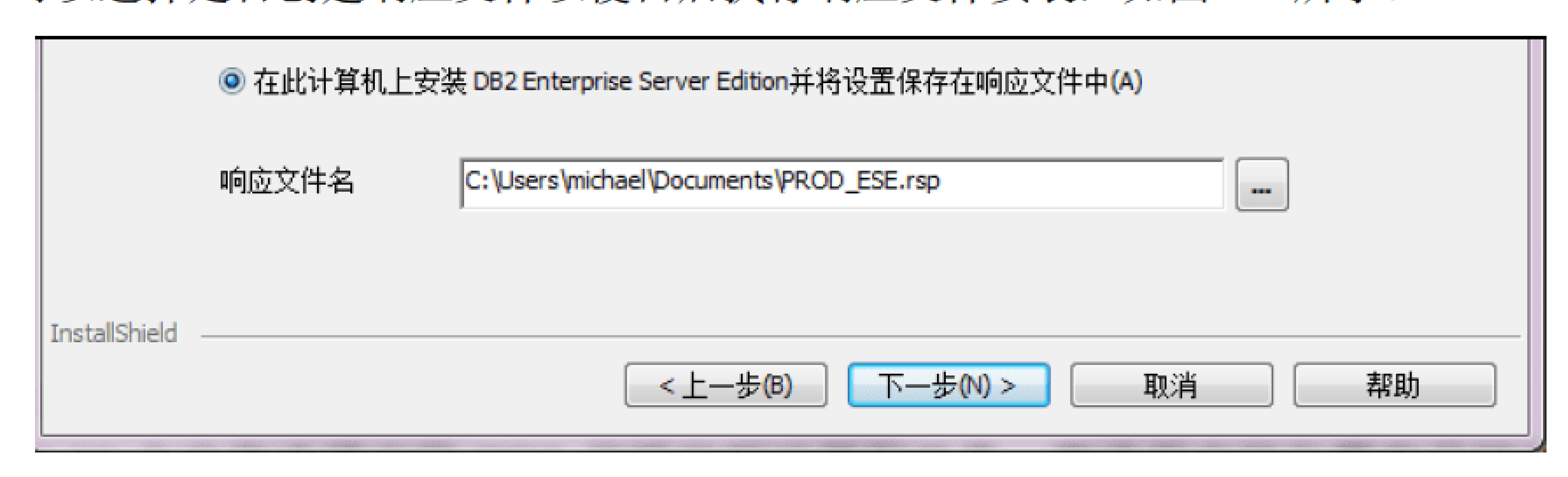


图 1-9　选择创建响应文件

(8) 选择安装文件夹。

图 1-10 允许选择安装 DB2 代码的驱动器和目录，要确保安装位置有足够的空间。对于这个示例，使用默认的驱动器和目录设置(如下所示)：



图 1-10　选择安装文件夹

驱动器：C:

目录：C:\Program Files\IBM\SQLLIB_01

单击“下一步”按钮继续。

(9) 配置 DB2 实例。

可以认为 DB2 实例是数据库的容器。必须有 DB2 实例，然后才能创建数据库。在 Windows 上进行安装时，会自动创建名为 DB2 的实例。后面第 2 章将详细讨论实例。

在默认情况下，DB2 实例监听端口 50000 上的 TCP/IP 连接，如图 1-11 所示。通过单击“协议”和“启动”按钮，可以分别修改默认的协议和端口。在这个示例中，建议使用默认设置。单击“下一步”按钮继续。



图 1-11　配置 DB2 实例

(10) 设置用户信息。

安装 DB2 之后，某些 DB2 进程会作为系统服务运行。为了运行这些服务，需要有操作系统账户。在 Windows 环境中，建议使用默认的 db2admin 用户账户，如图 1-12 所示。如果这个用户账户不存在，DB2 会在操作系统中创建(所以安装时需要管理员账号)。也可以指定使用现有的账户，但是这个账户必须具有本地管理员权限。在本书中，我们建议使用 ibmdb2 作为密码。单击“下一步”按钮继续。此时创建的用户 db2admin 具有最高权限，在 Windows 系统中，DB2 服务的启动都是以该用户的权限在执行。同时，该用户所属的组(默认为 db2admins)将作为 DB2 实例的最高管理权限组，拥有新创建实例的所有权限，如创建数据库、删除数据库、备份数据库等。属于该组的所有用户都可以执行这些动作，在默认安装下，该组只有 db2admin 一个用户。我们在后面的 DB2 数据库安全部分会详细解释这些。

(11) 启用操作系统安全性。

指定是否想对计算机上的 DB2 文件、文件夹、注册变量和其他对象启用操作系统安全性，如图 1-13 所示。



图 1-12　设置用户信息

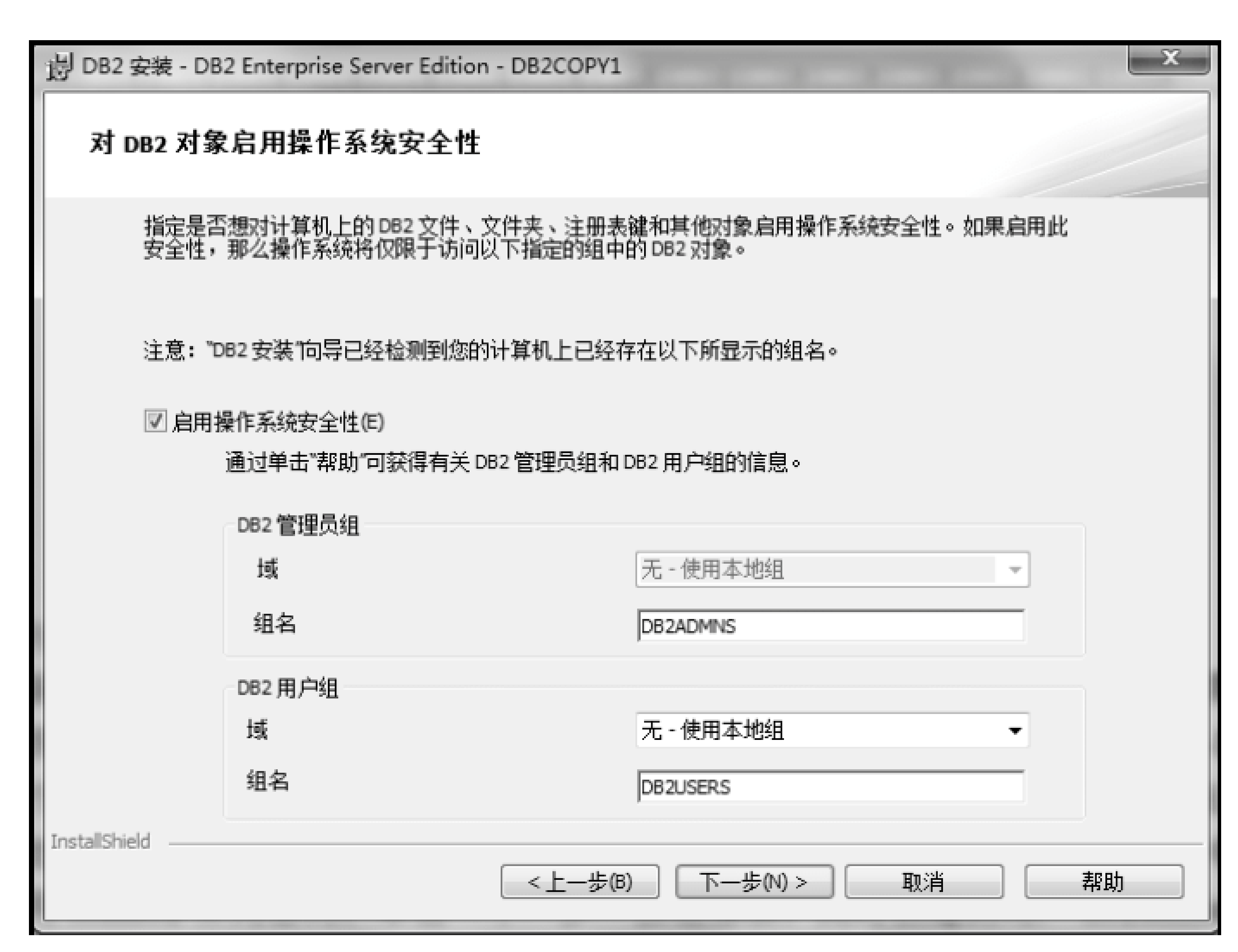


图 1-13　启动操作系统安全性

(12) 开始安装。

在经过前面一系列的选择后，我们开始安装 DB2 软件，如图 1-14 所示。

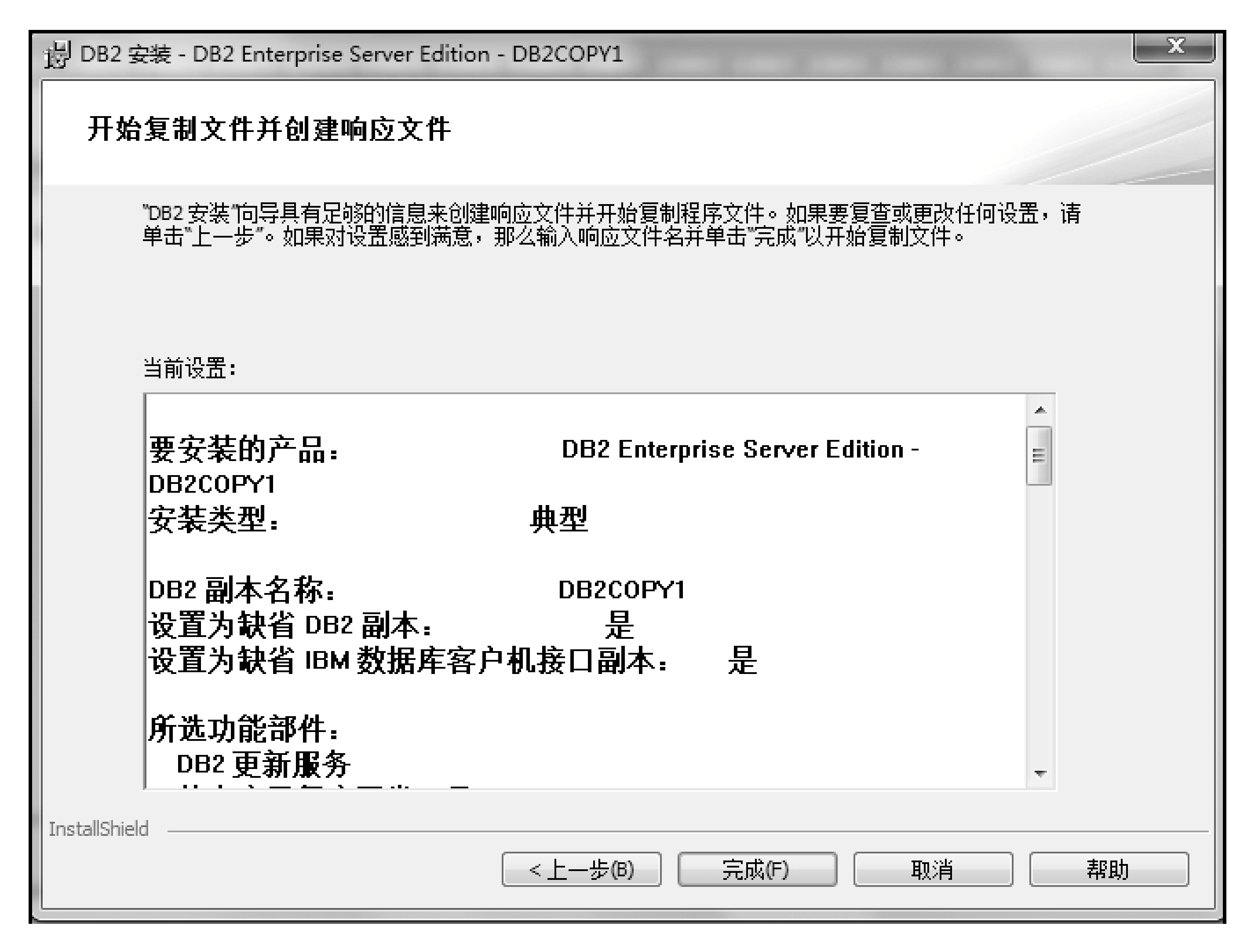


图 1-14　开始安装

(13) 完成安装，如图 1-15 所示。

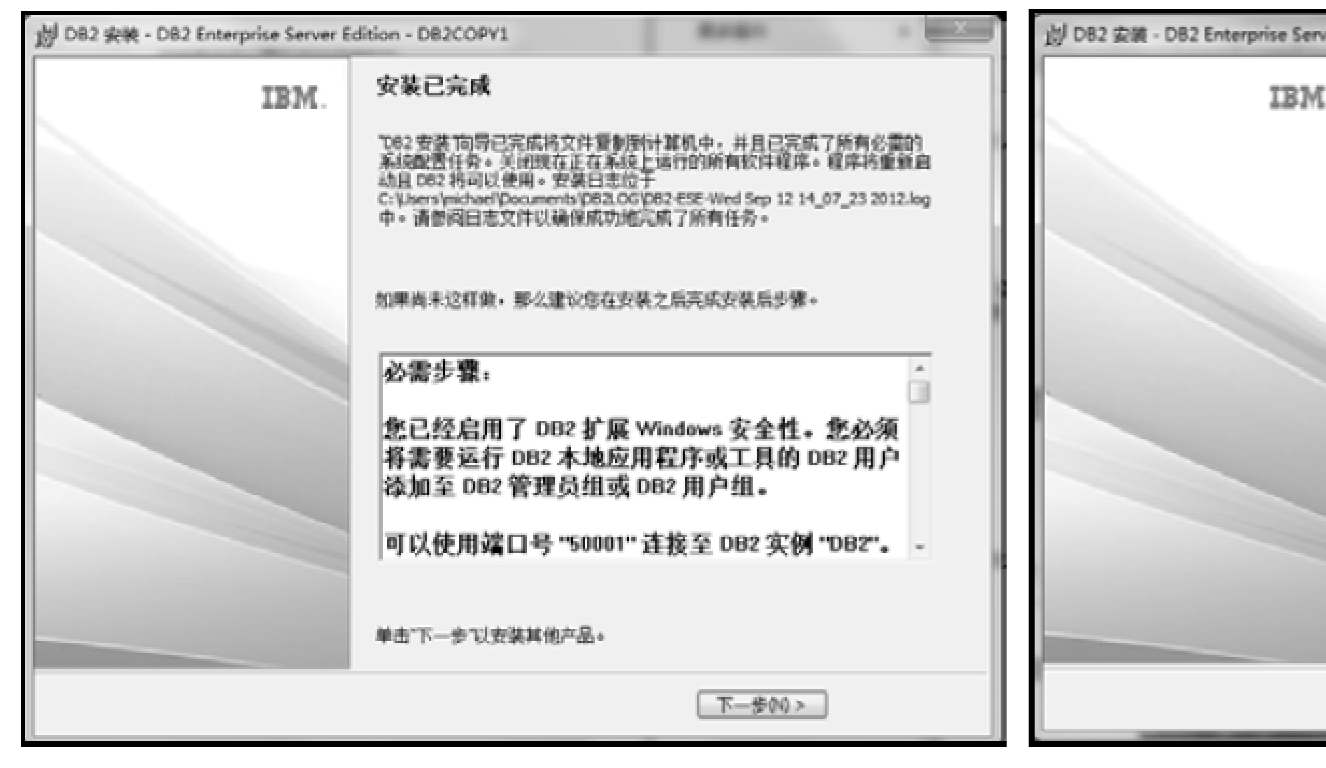




图 1-15　完成安装

(14) 验证安装。

在安装完成之后，显示称为“DB2 第一步”的启动面板(也可以用命令 db2fs 来启动)。

在“DB2 第一步”启动面板中，选择“创建数据库”选项卡，然后按照向导的指示创建 SAMPLE 数据库，如图 1-16 所示。



图 1-16　选择并创建 SAMPLE 数据库

选择“XML 和 SQL 对象和数据”选项并单击“确定”按钮，如图 1-17 所示。

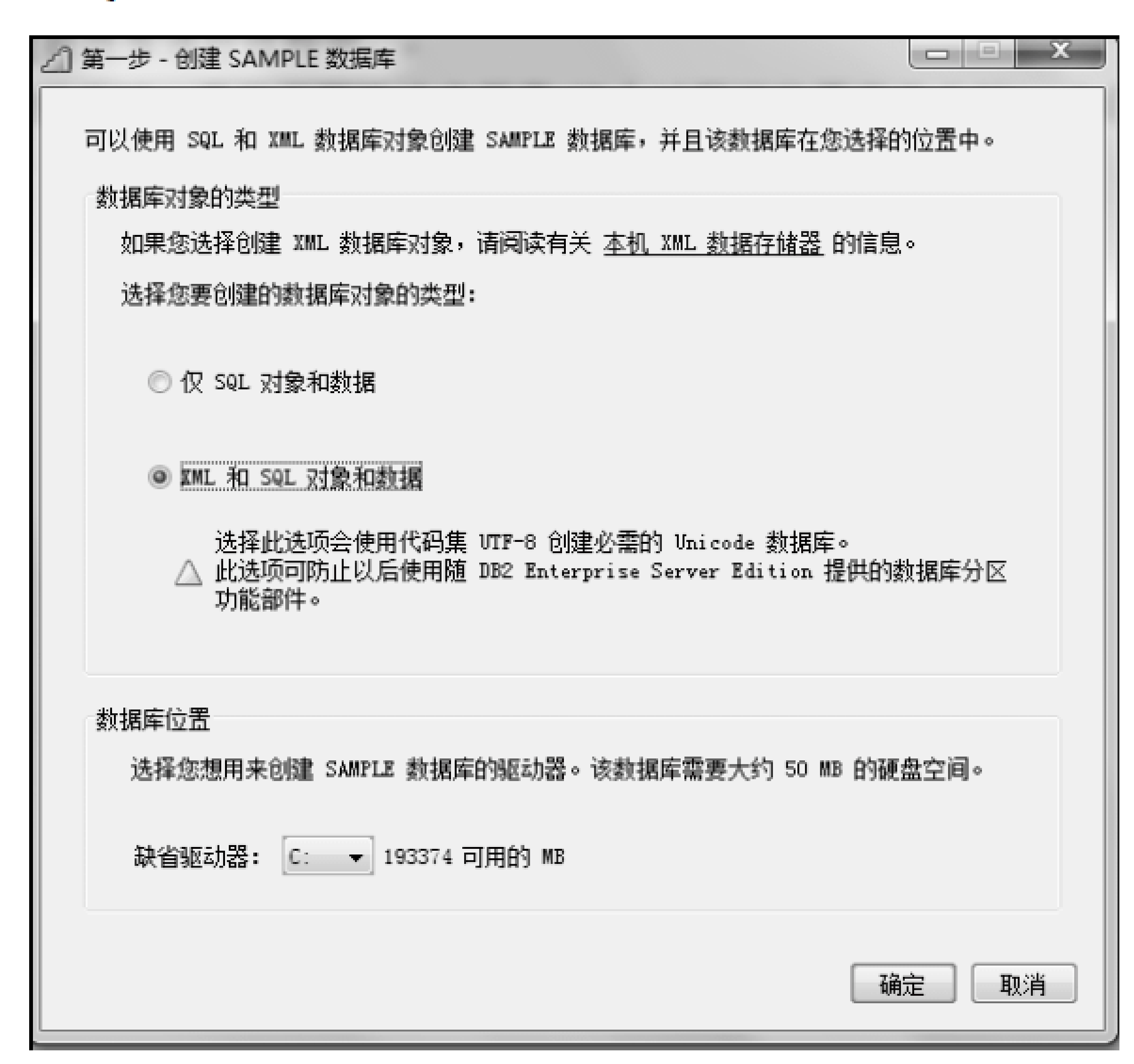


图 1-17　选择“XML 和 SQL 对象和数据”选项

(15) 在创建数据库时，会显示下面的进度屏幕(这个过程可能要花几分钟)，如图 1-18 所示。

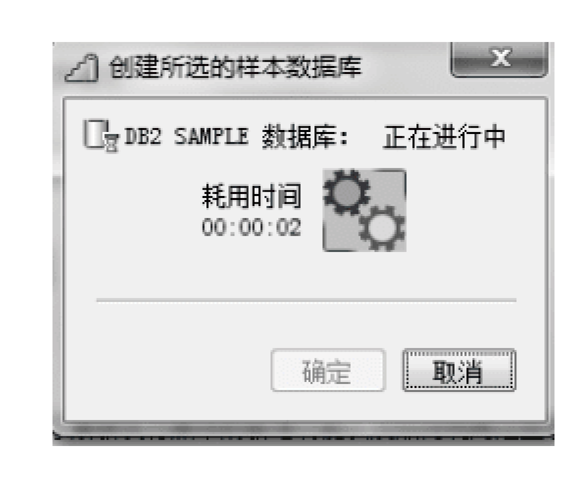


图 1-18　正在创建数据库

(16) 完成数据库的创建。

数据库创建完成后，打开控制中心，并检查左边面板中现在是否出现了名为 SAMPLE 的数据库，如果出现，就表示创建成功。可能必须刷新控制中心视图，才能看到新的变更。注意这个 SAMPLE 数据库，我们本书的所有练习和实验都是在这个数据库上实现的，所以建议初学者创建。

(17) 重新启动计算机，完成安装。

尽管正式的 DB2 安装文档中没有提到这个步骤，但是我们建议重新启动系统(如果可能的话，不重启也是可以的)，从而确保成功地启动所有进程并清理内存资源。这个步骤是可选的。

1.2.2　DB2 在 Linux/UNIX 上的安装

DB2 在 AIX、HP-UX、Sun Solaris 以及 Red Hat Linux 上的安装步骤基本类似，只是在安装前的准备工作上有差异。在这些平台上，如果调用 db2setup 安装向导，安装界面和 Windows 的图形界面基本上差不多，你需要配置好 JRE 运行环境和 X 环境。

建议大家使用 db2_install 命令行安装 DB2。而且建议读者学习的时候最好能在本地机器上利用 VMware 虚拟机安装 Linux 操作系统，然后在 Linux 上安装 DB2 数据库并在这个环境中学习和做实验。因为通常情况下，大多数比较重要的 DB2 数据库都是运行在 UNIX 环境中的，而 Linux 是最接近 UNIX 的环境。

如果要在 Linux/UNIX 上安装 DB2 数据库，一般需要注意以下几点：

- 使用 root 用户安装(DB2 V9 可以使用非 root 用户安装，但是有一些限制，建议读者还是使用 root 用户安装)。
- 确保硬件平台、内存和硬盘满足安装的最低要求。
- 确保正确地设置安装所需的内核参数。
- 确保操作系统版本补丁、操作系统内文件包和 DB2 的版本相兼容。

在正确设置好以上必需的环境和参数后，下面我们使用 db2_install 来安装 DB2。

安装步骤

db2_install 脚本会安装 DB2 产品中由你指定的所有组件，并具有字符界面支持，但并不执行用户和组创建、实例创建或配置。按照以下步骤开始使用 db2_install 进行安装：

以 root 用户登录。插入包含 DB2 软件的介质 DVD，或者访问存储安装映像的文件系统。改变目录至 <DVD_mount>/ese/disk1。

运行以下命令：

```
#./db2_install -b DB2DIR -p productName
```

其中，DB2DIR 是要安装 DB2 产品的路径，productName 是要安装的产品的名称。对于完整的 Enterprise Server Edition，选择 ESE；也可以选择客户机(CLIENT)或运行时客户机(RTCL)版本。

可以不为 db2_install 命令提供任何参数，在这种情况下会提示输入产品名称和安装路径。可以通过运行命令 db2_install –h 来了解详细的用法信息。如果没有使用 - l 选项指定日志路径，那么可以在/tmp 目录中找到安装日志文件。

安装信息如下：

```
<DVD_mount>/db2install/V9.7/ese/disk1 #./db2_install
Default directory for installation of products - /opt/ibm/db2/V9.7
***********************************************************
Do you want to choose a different directory to install [yes/no]
yes
Enter full path name for the install directory -
------------------------------------------------
/opt/ibm/db2/V9.7
Specify one or more of the following keywords,
separated by spaces, to installDB2products.
  CLIENT-----------------------客户机
  RTCL--------------------------运行时客户机
  ESE-----------------------------ESE 服务器
Enter "help" to redisplay product names.
Enter "quit" to exit.
***********************************************************
ESE
DB2 installation is being initialized.
Total number of tasks to be performed: 39
Total estimated time for all tasks to be performed: 853
```

在 Linux/UNIX 上完成安装后，DB2 的安装目录如图 1-19 所示。

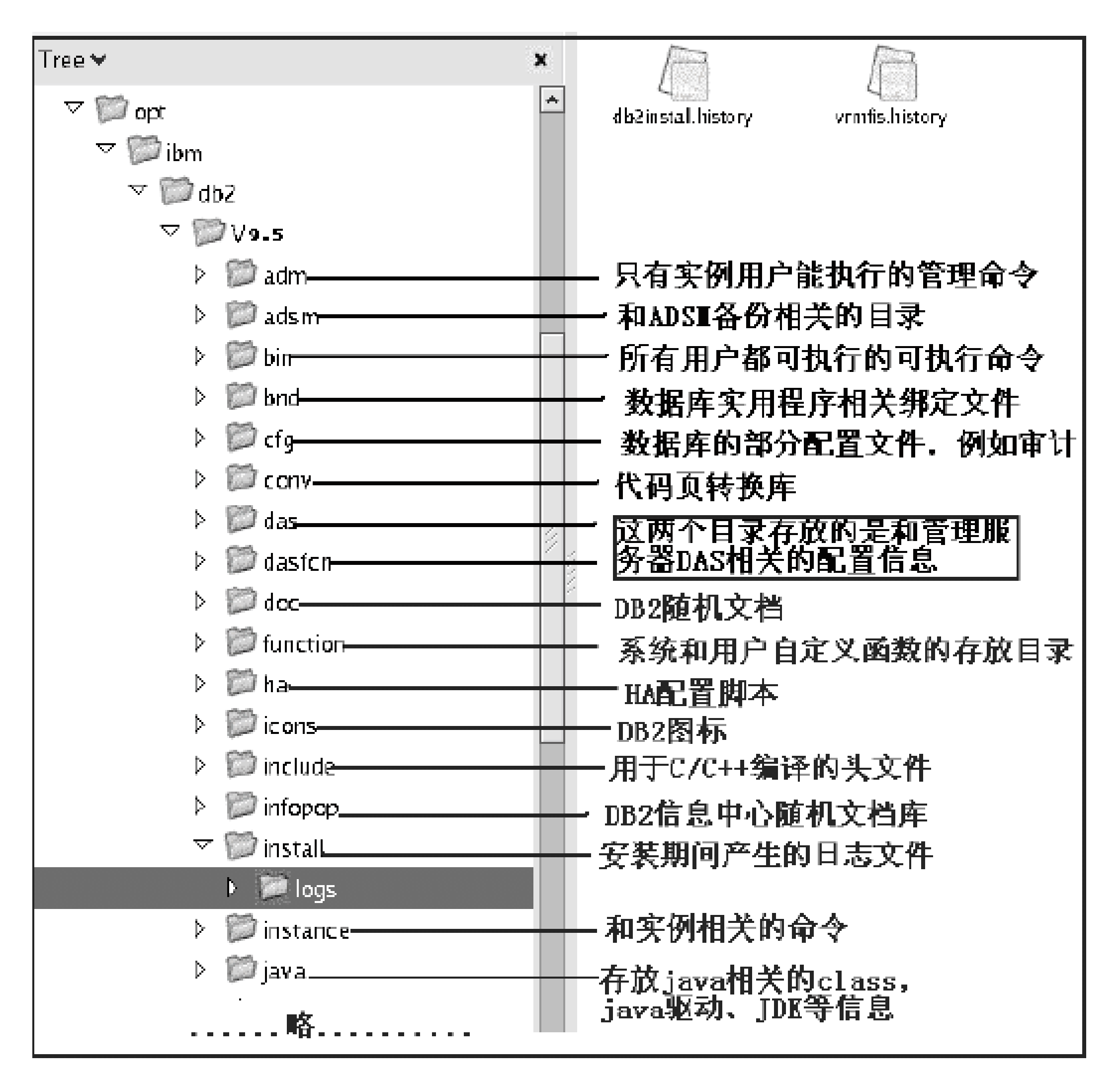


图 1-19　DB2 的安装目录

1.3　DB2 数据库的体系结构

安装完 DB2 软件后，在开始学习 DB2 之前，先简单了解一下 DB2 数据库的体系结构。图 1-20 显示了 DB2 V9 数据库对象的体系结构。

系统

DB2 体系结构中的最高一层是系统，一个系统表示 DB2 的一个安装。在由很多机器组成的网络环境中，我们有时也称系统为数据库分区。一个系统可以包含多个 DB2 实例，每个实例能够管理一个或多个数据库。

An instance

实例也称为数据库管理器(Database Management Application)，是数据库管理器在内存中的映像，是管理数据的 DB2 代码。实例相当于 Informix 的 Informix Server，在一台机器上可以有多个相互独立的实例，实例之间彼此独立，同时运行，不会相互影响。每个实例可以管理若干个数据库，一个数据库只属于一个实例。实例可控制对数据执行的操作，并管理分配给实例的系统资源。每个实例都是独立的运行环境，可以编目数据库和设置配置参数。可以在同一物理服务器上创建多个实例，并为每个实例提供唯一的数据库服务器

环境。

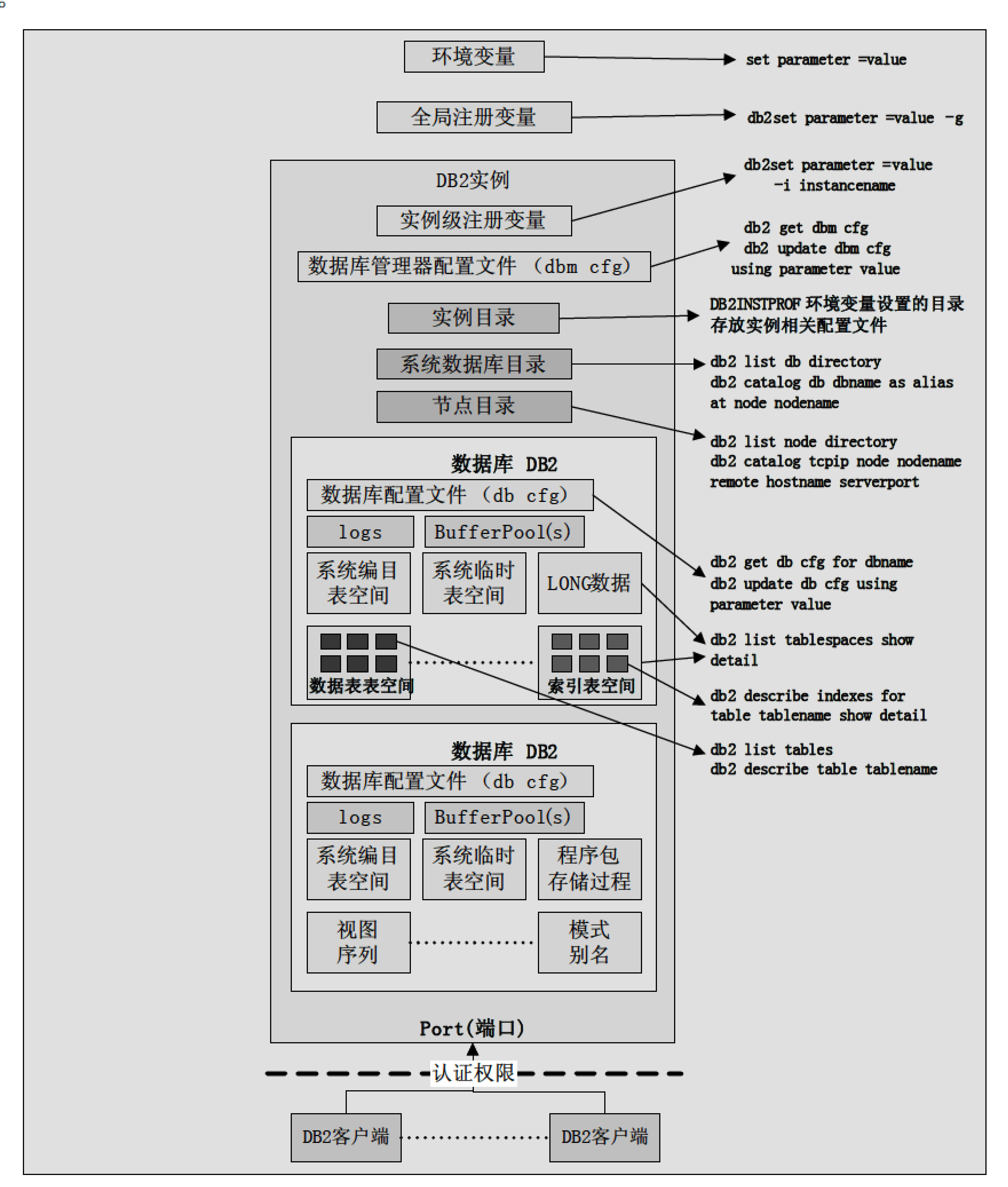


图 1-20　DB2 体系结构概览

关于如何创建实例、管理实例等详细内容，我们会在“第 2 章：创建实例和管理服务器”中详细讲解。

Configuration files and the DB2 profile registries

和许多其他关系数据库管理系统(RDBMS)一样，DB2 使用不同的配置参数来管理、监视和控制 DB2 系统的行为。这些机制包括如下内容。

环境变量

环境变量是在操作系统级别上定义的变量。例如，在 Windows 平台中，可以为变量创建新的项，也可以通过选择**“控制面板”-->“系统”-->“高级”-->“环境变量”**来编辑现有变量的值。在 UNIX 中，通常可以将安装了 DB2 之后提供的脚本 db2profile(Bourne 或 Korn shell)或 db2cshrc(C shell)添加到.login 或.profile UNIX 初始化文件中。db2profile/db2cshrc 文件包含了“export”UNIX 命令，这些命令能确保在数据库用户的环境中每次调用 shell 时都会传递 UNIX 环境变量的值。在创建实例后，我们需要配置相关环境变量来为用户提供运行环境，这部分内容会在“第 2 章：创建实例和管理服务器”中讲解。

DB2 概要文件注册变量

DB2 通过使用概要文件注册变量来集中控制环境变量，不同的概要文件提供了不同级别的支持。注册变量定义了 DB2 操作环境，这些变量存储在 DB2 注册文件中，具有两种注册变量。一种是全局变量，变量的设置是系统范围的。另一种是实例变量，变量的设置用于特定的实例。在实例上定义的变量值能够覆盖全局变量中同名变量的设置。

在单分区数据库中，有 3 类概要文件注册变量：

- DB2 实例级概要文件注册变量：DB2 变量的大多数放在这个注册文件中。特定实例的变量设置放在这个注册文件中。可以在全局级(db2set -g 选项)和实例级(db2set -i 选项)设置 DB2 概要文件注册变量。在实例层定义的值能够覆盖全局层中同名变量的设置。
- DB2 实例概要文件注册表：此注册表包含系统可识别的所有实例名的列表。可通过运行 db2ilist 命令来查看系统中提供的所有实例的完整列表。
- 环境变量：这类变量的设置方法因操作系统的不同而有所差异。

系统启动后，在检查系统的变量时，**按照先环境变量，再全局级注册变量，最后实例级注册变量的顺序来搜索**。关于这部分变量的设置在第 2 章有详细讲解。

配置参数

配置参数是在两个不同的级别(实例级和数据库级)上定义的。每个级别上的变量都是不同的(不像注册变量那样可以在不同级别上定义相同的变量)。

配置参数能够影响数据库或数据库管理员的操作特性，它们存储在配置文件中。数据库管理器配置参数是在 DB2 实例创建的时候创建的，其中包含的参数能够在实例层、独立于实例的任何数据库影响系统资源。

许多参数都会被数据库设置为默认值，管理员可以根据自己的需要修改它们。数据库的配置文件是在数据库创建的时候创建的，位于数据库所在的目录。每个数据库都有配置文件。数据库的配置参数定义了分配给数据库的各种资源的数量。许多的配置参数值可以被修改，以满足数据库运行的各种需求，如提高性能和容量等。不同参数的修改依赖于特定数据库应用的需求。

关于配置参数的阐述，我们分别在“第 2 章：创建实例和管理服务器”和“第 3 章：创建数据库和表空间”中讲解。

Databases

关系数据库使用一组表来管理数据，表由在行和列中以逻辑关系排列的数据组成，每个表的数据在逻辑上相关，在表之间能够定义关系。

每个数据库包含一组系统编目表(或称之为数据字典)、配置文件和恢复日志，系统编目表用于描述数据的逻辑和物理结构，配置文件包含所有为数据库分配的配置参数值，恢复日志记录正在进行的事务处理和可存档的事务处理。

数据库可以是本地的，也可以是远程的。本地数据库物理上位于本地的机器上；当数据库物理上驻留在另一台机器上时，则称为远程的。关于数据库的详细讲解请参见“第 3 章：创建数据库和表空间”。

Tablespaces

表空间是数据库中表数据与数据库之间的逻辑中间层，数据库中的物理空间组织为表空间的集合，而表空间是表的逻辑集合。表空间包含容器集合，容器是用来描述物理空间分配的一般术语。数据库将数据存储在自己的表空间容器中。

表存储在一个或几个表空间中，为了提高性能，或者为了便于表空间的备份，可以将表中不同类型的数据分别存储在不同的表空间中，比如将常规数据存储在第一个表空间中，将表的索引存储在第二个表空间中，将大对象数据存储在第三个表空间中。

表空间最终会映射到物理存储介质上，对物理存储的合理使用可以让管理员有效地控制数据库的性能。例如，可以使用最快的设备或内存硬盘来存储频繁使用的表，使用较慢的设备存储不经常使用的数据。表空间的概念提供了对底层存储物理设备的更加灵活的使用。

表空间的规划设计会显著影响数据库运行的性能。关于表空间的详细规划设计，我们会在“第 3 章：创建数据库和表空间”中详细讲解。

Connectivity and DB2 directories

节点目录

节点目录用于存储远程数据库的所有连通性信息。在节点目录中记录与每个系统进行通信所需要的信息：例如机器(其中包含了你想连接的数据库)的主机名或 IP 地址，还有相关的 DB2 实例的端口号和使用的通信协议。要想得到你想要连接的远程实例的端口号，可以通过查看远程实例的 dbm cfg 中的 svcename 参数来实现。该值通常对应于 TCP/IP services 文件中的某一项。节点目录主要用于配置客户端到服务器的通信。

系统数据库目录(或系统 db 目录)

系统数据库目录包含本地数据库目录和从远程机器上映射到本地的数据库目录，它是我们访问数据库的一个入口，连接数据库时要首先去系统数据库目录中判断这个数据库，是否存在，然后再判断这个数据库是本地数据库还是远程数据库。如果是本地数据库，就直接到本地物理目录上访问；如果是远程数据库，那么还要找到这个远程数据库在哪个节点上，然后再到节点目录中找到这个节点的通信信息，最后需要连接到远程的节点上来访问这个远程的数据库。

本地数据库目录(或本地 db 目录)

本地数据库目录包含了有关本地数据库(即驻留在你目前正在使用的机器上的数据库)的信息。当使用 create database 命令创建数据库时，在该目录中会添加一个条目。

关于节点目录、系统数据库目录和本地数据库目录的详细信息，会在“第 4 章：访问数据库”中详细讲解。

模式

模式是数据库对象的逻辑分组集合，细化了数据库的“粒度”，帮助对表和其他数据库对象进行逻辑分组。模式可以归个人所有，拥有者可以控制对数据以及其中对象的存取。

模式是数据库对象特征划分的结果集，可以表示数据库对象集的特点，有一定的安全作用。数据库中所建的每一个对象都有模式，这些模式会隐式或显式地增加为对象的前缀。创建用户时，系统会为每个用户默认隐含建立与用户名同名的模式名。当创建数据库中的对象时，如果写明了它的模式名(即对象的前缀)，那么此模式即为该对象的模式；如果未指明模式名，那么与当前用户名同名的模式即为当前对象的模式。关于模式的创建会在“第

5 章：创建数据库对象”中讲解。

Tables, indexes and large objects

表是数据库的基本组成单元，是对客观世界中实体的一种描述。表由行、列组成。表的每列描述了对应实体的一个属性，同一列的数据都具有相同的数据类型。表的每一行都描述了一个实体的信息。所有数据库和表数据都被存储在表空间中，表中的数据在逻辑上是相关的。可定义表之间的关系。索引是与表相关的有序指针集，用于性能目的并确保唯一性。视频、音频和扫描文档等可以作为大对象(LOB)存储在数据库中。表、索引和 LOB 驻留在表空间中。为了提高性能，可以把表数据、索引数据和大对象数据分别存放到不同的表空间，关于这部分的详细讲解，请参见“第 5 章：创建数据库对象”。

视图

视图是高效率的数据操纵机制。视图是“虚拟”的表，视图不是真正的表，不需要永久性存储器，它的数据本质上还是来自于数据库中的基表。

视图是从一个或几个基本表导出的表，也可从其他视图导出。某一用户可以定义一个或多个视图，经授权后一个视图也可为多个用户共享，这一点与基表类似。视图是虚表，而基表是实表。虚表只有定义，没有对应的物理数据；而实表既有定义，又有对应的物理数据。虽然如此，视图一经定义就可和基表一样被查询、删除，还可用来定义新的视图。但更新(增、删、改)视图的操作有一定限制。视图给用户和应用程序提供了一种灵活的操纵数据的方式。

关于视图的详细讲解，请参见“第 5 章：创建数据库对象”。

Logs

日志是用于恢复目的的文件。日志记录了对数据库进行的每个 SQL 操作。万一发生故障，要将数据库恢复到某个一致的时间点，日志就显得至关重要了。数据库恢复日志保存对数据库所做的全部更改的记录，包括对新表的添加或对现存表的更新。这个日志由大量日志块组成，每一个日志块包含在名为日志文件的单独文件中。

数据库恢复日志可以用于确保故障(例如系统断电或应用程序出错)时不会使数据库处于不一致的状态。如果发生故障，就回滚已进行但未落实的更改，重新执行可能没有实际写入磁盘的所有已落实的事务。这些操作确保了数据库的完整性。关于使用数据库日志、备份和恢复的详细讲解，请参见“第 7 章：数据库备份与恢复”。

Bufferpool(s)

缓冲池是一块内存区域，所有索引和数据页(除了 LOB)都必须有序地经过该区域，从

而进行处理。缓冲池是数据库管理器使用的主要高速缓存。在数据库性能问题方面，缓冲池是进行调优的最重要对象。关于缓冲池的规划设计，请参见“第 3 章：创建数据库和表空间”。

系统编目表

每个数据库都会创建和维护一组描述数据的逻辑和物理结构的系统编目表。这些表包含有关数据库对象(表、视图、索引以及约束和数据库授权)定义的信息，以及用户对这些对象所拥有权限的安全性信息。这些表存储在 SYSCATSPACE 表空间中。它们在数据库创建时被创建，当创建、修改或删除对象的时候，DB2 插入、更新或删除描述对象的目录行。系统编目表是只读类型，因为它们是被 DB2 维护的。不能显式地创建或卸载它们，但是可以使用目录视图查询它们的内容。关于系统编目表的详细讲解，请参见“第 3 章：创建数据库和表空间”。

程序包

程序包是在准备包含编译以后的 SQL 语句和控制结构的程序时，所产生的对象，这些 SQL 语句和控制结构存在于单个源文件中，并且在运行中使用。程序包由段(section)组成。段包括编译以后的 SQL 语句。虽然每个段对应于单条语句，但是并非每条语句都具有段。程序包和应用开发相关，本书中没有涉及程序包的讲解。

性能视图

性能视图用于存储数据库管理员的性能和操作信息，应用程序可以使用这些信息。这些信息对于调试性能、诊断问题是非常有用的。性能视图可以用于提取数据库管理器特定对象和对象组的当前活动。性能视图存放在 DB2 的系统编目表空间中，关于这部分的详细信息，会在“第 9 章：DB2 基本监控方法”中详细讲解。

诊断文件

每个实例都有诊断文件，当数据库出现问题时，首先要检查诊断文件来判断数据库出现了什么类型的错误，关于数据库诊断的详细信息，会在《高级进阶 DB2(第 2 版)》的“第 8 章：DB2 故障诊断”中详细讲解。

认证和权限

认证提供了一种用户验证机制，决定用户和密码能否连接实例或数据库。而权限决定用户能否合法地存取数据。数据库相关的权限存储在数据库系统编目表中。关于这部分的详细信息，会在《高级进阶 DB2(第 2 版)》的“第 9 章：数据库安全”中详细讲解。



创建实例和管理服务器

在安装完 DB2 数据库后，首先需要做的就是创建实例。因为要创建数据库，就必须先创建实例，数据库是运行在实例之上的。在本章中，向大家讲解如何创建实例以及与实例相关的命令。DAS(Database Administrator Server)是一种特殊类型的实例，主要执行远程管理任务。

本章主要讲解如下内容：

- 实例
- DAS(管理服务器)

2.1 实例

2.1.1 实例的概念

从 DB2 体系结构的方面来看，实例实际上就是 DB2 的执行代码和数据库对象的中间逻辑层。实例可以看成关于所有的数据库及其对象的逻辑集合，也可认为是所有的数据库及其对象和 DB2 代码之间的联系和结合。实例为数据库运行提供环境。数据库在运行时，实例用来为数据库提供安全、通信、内存分配和进程间通信等功能。这样一来，数据库只负责前台正常的运行，而一些后台的事情由实例来进行管理。实例对用户和开发人员来说是透明的。实例本质上由一组后台进程和共享内存组成。实例和数据库不一样的地方是，数据库是物理的，我们的表、索引存放在数据库中是要占用物理存储的；而实例是逻辑的，是共享内存、进程和一些配置文件(实例目录)的集合。当实例停止时，共享内存释放，进程停止。实例就相当于 Windows 中服务的概念。所以大家在学习 DB2 时，首先要搞清楚实例的概念。

在实际生产系统中，我们可能需要创建多个实例来执行下列操作：

- 将一个实例用于开发环境，而将另一个实例用于生产环境
- 为特定环境调整实例
- 限制存取机密信息
- 控制为每个实例指定 SYSADM、SYSCTRL、SYSMAINT 和 SYSMON 权限
- 优化每个实例的数据库管理器配置
- 限制实例故障的影响。如果发生实例故障，那么只有一个实例受影响，其他实例可以继续正常工作

当然，系统中的实例不是越多越好，如果在系统中创建的实例过多，不仅会造成额外的资源消耗(内存、硬盘空间等)，还会增大管理开销。过多的实例数量可能会有下面的影响：

- 每个实例都需要额外的系统资源(虚拟内存和磁盘空间)
- 由于要管理其他的实例，因此增加了管理工作量
- 更多管理任务，因为要管理附加实例

注意：

在许多数据库产品中都有类似实例的这个概念，例如在 Oracle 中也叫实例(instance)，在 Informix 数据库中有 Server 的概念，Sybase 和 SQL Server 中的 Server 概念也和实例类似。

2.1.2 创建实例

在 Windows 上 DB2 的安装过程中，如果没有其他实例的名称为“DB2”，那么将自动创建名为 DB2 的数据库管理器初始实例，该实例由 DB2INSTANCE 环境变量定义。如果安装了 DB2 V8，并且已升级到 V9.1 或 V9.7，那么默认实例为“DB2_01”。在 Windows 上创建实例的时候，不需要创建用户，创建完实例后，实例会作为服务存在。

而在 Linux/UNIX 上，要想创建实例，就必须首先创建和实例名一样的用户及该用户所属的组。之所以需要创建用户，主要是因为需要以该用户的 home 目录作为实例目录，存放实例相关的实例目录结构。

实例可以在 DB2 向导安装期间创建，但业务需求可能需要我们手工创建其他实例。创建实例需要使用 db2icrt(db2 Instance CReaTe)命令，db2icrt 命令的语法如下所示：

```
For Linux and UNIX

>>-db2icrt--+-----+--+-----+--+---------------+----------------->
```

```
            +- -h-+  '- -d-'  '- -a--AuthType-'
            '- -?-'
>--+---------------+--+---------------+--+---------------+------>
   '- -p--PortName-'  '- -s--InstType-'  '- -u--FencedID-'
>--InstName----------------------------------------------------><

For Windows

>>-db2icrt--InstName--+---------------+------------------------>
                      '- -s--InstType-'
>--+------------------------+--+-------------------+----------->
   '- -u--UserName, Password-'  '- -p--InstProfPath-'
>--+---------------+--+---------------+------------------------>
   '- -h--HostName-'  '- -r--PortRange-'
>--+-----------------------------------------------------+----->
   '- -j--"TEXT SEARCH-+--------------+-+-------------+-"-'
                       '-,servicename-' '-,portnumber-'
>--+-----+-----------------------------------------------------><
   '- -?-'
```

有一点我们需要特别注意，在 Linux 和 UNIX 上创建实例时，必须有和实例同名的用户存在。如果该用户不存在，那么创建实例会报错而无法创建。如果用户存在，确保该用户未被锁定并且密码未到期。

注意：

在 Linux 和 UNIX 上创建实例时，必须创建和实例名一样的用户，而在 Windows 上不需要创建和实例同名的用户，但要确保创建的实例名与存在的服务名不相同，否则无法创建。

使用 db2icrt 命令创建实例的步骤如下：

(1) 使用 root 权限登录(在 Windows 上使用系统管理员账号登录)。

(2) 首先利用操作系统命令(mkuser、useradd；mkgroup、groupadd)创建实例的用户和组，一般来说我们需要创建表 2-1 所示的用户和组并设置密码。表 2-1 中的用户名和组名是使用 DB2 安装向导期间由 DB2 默认生成的。实际生产中，我们可以根据自己需要创建自己特定的组名和用户名，此处仅为说明作用。

表 2-1　默认生成的用户和组

用　　户	示例用户名	示 例 组 名
实例所有者	db2inst1	db2iadm1
受防护的用户	db2fenc1	db2fadm1

- 实例所有者的 home 目录(假如为/home/db2inst1)是将在其中创建实例目录的位置。
- 受防护的用户(db2fenc1)用于在 DB2 数据库所使用的地址空间之外运行用户定义的函数(UDF)和存储过程。这个用户和应用开发有关，通常没有什么用，但是作为创建实例却是必需的。很多初学者往往被这个用户迷惑，其实不创建这个用户，也可以使用实例用户作为受防护的用户。但是从应用程序安全和维护角度而言，建议创建这个用户。

(3) 运行 db2icrt 命令。例如，在 Linux 或 UNIX 操作系统上：

```
DB2DIR/instance/db2icrt -a AuthType -u FencedID InstName
```

在 Windows 操作系统上：

```
DB2DIR\bin\db2icrt InstName
```

其中，DB2DIR 是 DB2 安装目录。在 Linux/UNIX 操作系统上，默认的 DB2 安装目录是/opt/IBM/db2/V9.7。

-a AuthType(Linux 或 UNIX)

表示实例的认证类型。AuthType 可为 SERVER、CLIENT、SERVER_ENCRYPT 和 DCS_ENCRYPT 其中之一。SERVER 是默认值。此参数是可选的，这些认证类型和安全有关，关于这部分的详细内容，在《高级进阶 DB2(第 2 版)》的“第 9 章：数据库安全”中会深入讲解。

-u FencedID

表示将用来运行受防护用户定义的函数(UDF)和受防护存储过程的用户的名称。这个用户和应用开发有关，虽然通常用不到，但对于创建实例是必需的。

InstName

表示实例的名称。实例的名称必须与拥有实例的用户的名称相同。指定创建的拥有实例的用户的名称。在拥有实例的用户的主目录中创建该实例。

例如，如果正在使用服务器认证，受防护用户为 db2fenc1，并且拥有实例的用户为 db2inst1，那么使用以下命令以在 AIX 系统上创建实例：

```
/opt/IBM/db2/V9.7/instance/db2icrt -a server -u db2fenc1 db2inst1
```

db2icrt 命令除了上述必需选项外，还有一些可选选项，如下所示：

- –s　指定所创建实例的类型，有以下几种类型：
  - ◊ ese　用于创建具有 DPF 支持的 DB2 数据库服务器的实例，DB2 数据库服务器

带有本地和远程客户机。此类型是 DB2 企业服务器版的默认实例类型。
- ◊ wse　用于创建 DB2 数据库服务器的实例，DB2 数据库服务器带有本地和远程客户机。此类型是 DB2 工作组版、DB2 易捷版或 Express-C 版的默认实例类型。
- ◊ standalone　用于创建 DB2 数据库服务器的实例，DB2 数据库服务器带有本地客户机。此类型是 DB2 个人版的默认实例类型。
- ◊ client　用于创建 IBM 数据库服务器客户机的实例。

- –p　如果想要在不同于 DB2PATH 的路径中创建目录，那么输入 db2icrt 命令之前必须设置环境变量 DB2INSTPROF。
- –u　用于指定 DB2 服务的账户名和密码，创建 ese 实例时需要此选项。
- –h　用于覆盖默认 TCP/IP 主机名。在创建默认节点(节点 0)时，将使用 TCP/IP 主机名。
- –r　用于指定当在 MPP(数据库分区)方式下运行时，分区数据库实例要使用的一系列 TCP/IP 端口。如果指定此选项，那么本地机器的 services 文件将使用下列条目进行更新：

```
DB2_InstName         baseport/tcp
DB2_InstName_1       baseport+1/tcp
DB2_InstName_2       baseport+2/tcp
DB2_InstName_END     endport/tcp
```

在 DB2 V10 版本中，db2icrt 命令会多出一些 DB2 pureScale 选项，我们将在后续的章节中详细介绍。

2.1.3　实例目录

实例创建后，会生成实例目录，实例目录存储着与数据库实例相关的所有信息。实例目录一旦创建，就不能更改其位置。在 Linux/UNIX 中为了拥有实例目录，必须创建和实例名相同的用户，其最终目的是为了以这个用户的 home 目录作为实例目录。

实例目录包含：

- 数据库管理器配置文件(db2systm)
- 系统数据库目录(SQLDBDIR)
- 节点目录(SQLNODIR)
- 节点配置文件(db2nodes.cfg)，db2nodes.cfg 文件用来定义参与 DB2 实例的数据库分区服务器。如果想要将高速互联用于数据库分区服务器通信，那么还可以使用 db2nodes.cfg 文件来指定高速互联的 IP 地址或主机名。

● 诊断文件、数据库错误日志等。

在 Linux/UNIX 操作系统上，实例目录位于 INSTHOME/sqllib 目录中，其中 INSTHOME 是实例所有者的主目录，图 2-1 是实例 db2inst1 的实例目录。

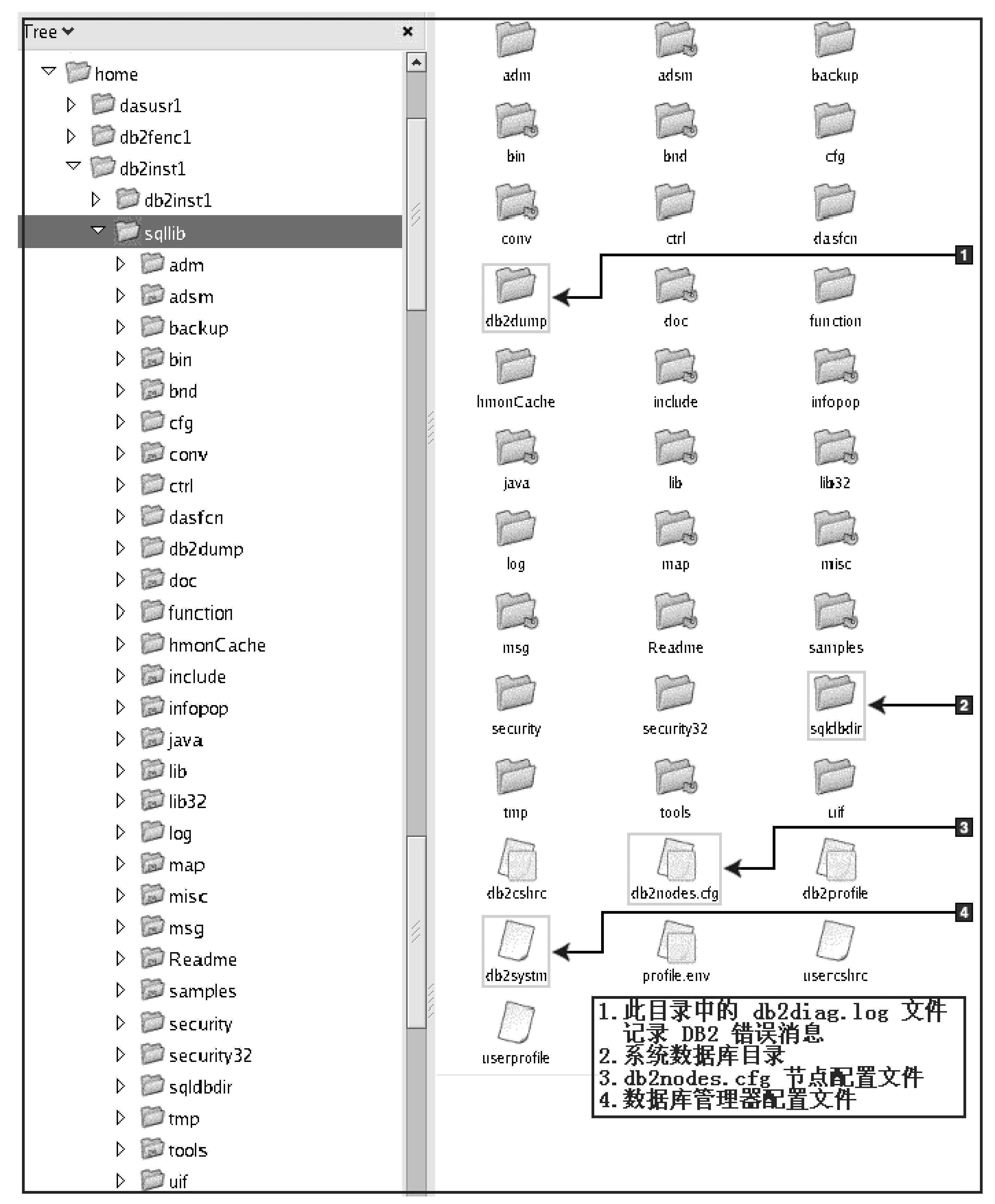


图 2-1　实例 db2inst1 的实例目录

在 Windows 操作系统上，实例目录位于安装了 DB2 数据库产品的目录下。实例名与服务名相同，因此应该不会发生冲突。实例名不应与别的服务名相同。你必须有创建服务所需的正确权限。可在 DB2PATH 中使用 DB2INSTPROF 环境变量更改实例目录的位置，这需要实例目录的写访问权。如果想要在不同于 DB2PATH 的路径中创建目录，那么输入 db2icrt 命令之前必须设置 DB2INSTPROF。

实例目录非常重要，下面我们举例来说明实例目录。在讲这个例子之前，我们先讲解一下 db2set 命令，之所以讲解这个命令是因为我们特殊定制实例时，需要用到这个命令。我们都知道，在操作系统中可以使用 set、setenv 或 export 命令来修改操作系统的环境变量。同样 DB2 实例本身也有实例级别的注册变量，为了修改这些默认的变量，我们使用 db2set 命令——在 set 前加上 db2 以表示设置 DB2 级别的变量。这个命令使用起来很简单。

要查看已经设置的注册变量，请从命令行执行下面这个命令：

```
db2set -all
```

你可能会得到类似下面这样的输出：

```
    C:\>db2set -all
    [e] DB2PATH=C:\Program Files\IBM\SQLLIB
    [i] DB2INSTPROF=C:\DOCUMENTS AND SETTINGS\ALL USERS\APPLICATION DATA\IBM\
DB2COPY1
    [g] DB2ADMINSERVER=DB2DAS00
```

正如你可能已经猜测到的那样，[i]表明该变量是在实例级别定义的，而[g]表明是在全局级别为系统中所有实例定义的；[e]表示是操作系统级别的环境变量。

要查看可以在 DB2 中进行定义的所有注册变量，请使用下面这个命令：

```
db2set -lr
```

要在全局级设置特定变量(在下面这个示例中为 DB2INSTPROF)的值，请使用：

```
db2set DB2INSTPROF="C:\INSTDIR" -g
```

要在实例级为实例“DB2”设置变量，请使用：

```
db2set DB2INSTPROF="C:\MY FILES\SQLLIB" -i DB2
```

请注意上面的示例，在两个级别(实例级和全局级)设置了同一个变量。当同一个注册变量在不同级别上进行定义时，DB2 总是会选择最低级别的值；在本例中，将选择实例级的值。

注意：

db2set 命令的等号(=)前后不该留有空格。某些注册变量为了使更改生效，要求停止和启动实例(db2stop/db2start，这两个命令后面会讲解)。另一些注册变量则没有这个要求。为了安全起见，建议在对注册变量作出更改后总是停止和启动实例。

现在你已经知道 db2set 命令的用法，下面首先设置 DB2INSTPROF 注册变量，然后在 Windows 上创建一个新的实例并查看这个实例的实例目录：

```
    C:\>db2set -all
    [e] DB2PATH=C:\Program Files\IBM\SQLLIB
    [i] DB2INSTPROF=C:\DOCUMENTS AND SETTINGS\ALL USERS\APPLICATIONDATA\IBM\
DB2COPY1
    ...............略
    C:\>db2set DB2INSTPROF=C:\INSTDIR  --重新设置 DB2INSTPROF 注册变量
    C:\>db2icrt prod------------------创建 prod 实例
    DB20000I  DB2ICRT 命令成功完成
```

在创建完 prod 实例后，我们发现在 DB2INSTPROF 目录下生成了一个和实例同名的目录。这就是实例 PROD 的实例目录，如图 2-2 所示。

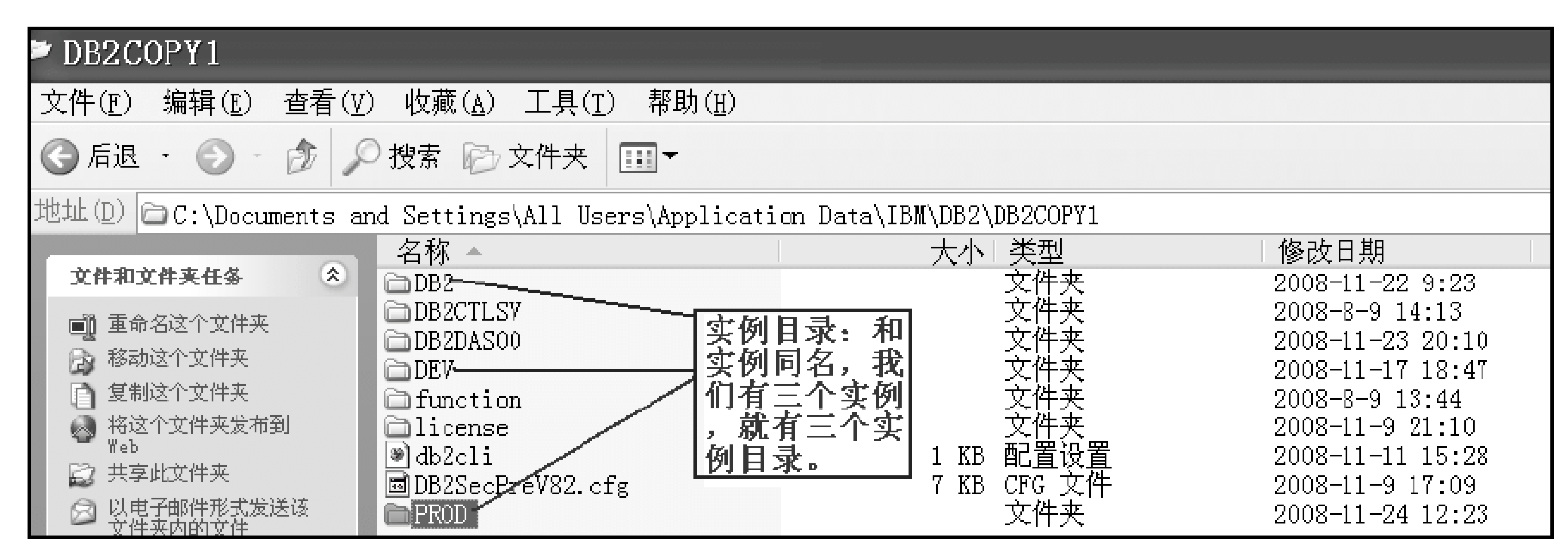


图 2-2　实例 PROD 的实例目录

进入刚刚创建的实例 PROD 的目录，如图 2-3 所示。

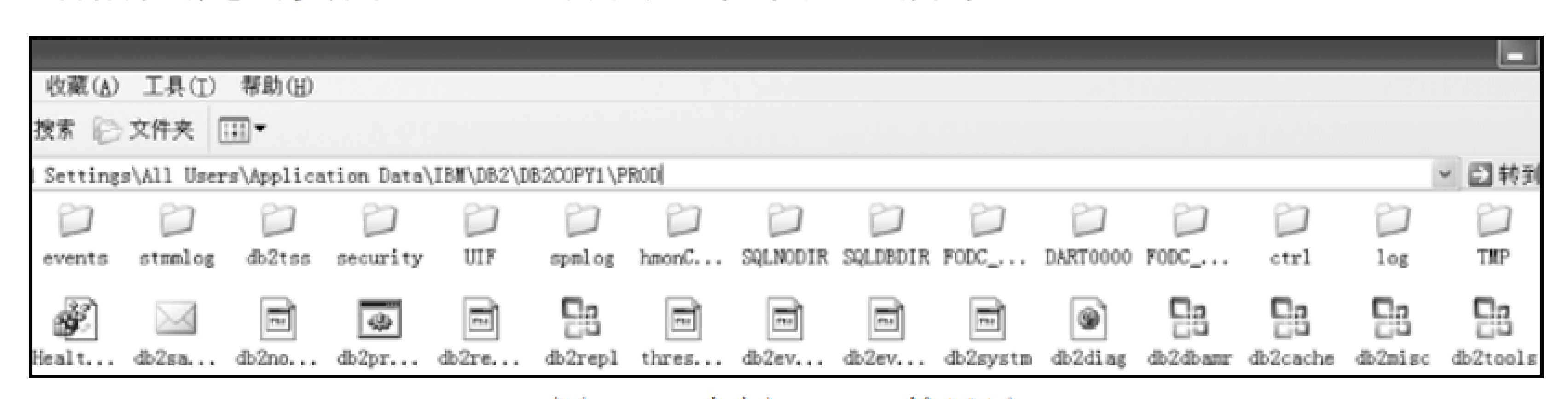


图 2-3　实例 PROD 的目录

在这个目录中存放实例的数据库管理器配置文件(db2systm)、系统数据库目录(SQLDBDIR)、节点目录(SQLNODIR)、节点配置文件(db2nodes.cfg)、诊断文件、数据库错误日志、安全配置等重要信息。所以实例目录至关重要。

同时，PROD 实例创建后，我们可以发现在 Windows 的服务面板中，多了一项

DB2COPY1-PROD 服务，如图 2-4 所示。

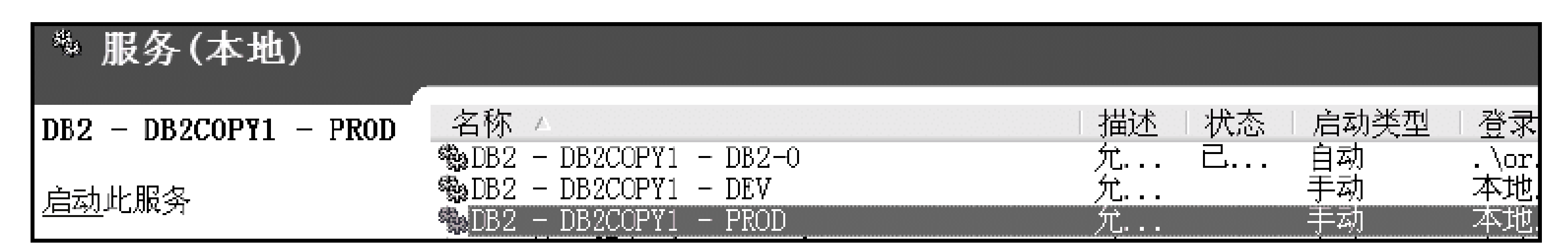


图 2-4　Windows 服务面板

这是因为在 Windows 上实例是作为服务存在的，而在 Linux/UNIX 上，实例是作为一组后台进程存在的。

在 Linux/UNIX 上可以通过 db2_ps 或 ps –ef|grep –i INSTNAME 查看 DB2 进程的状态：

```
    DS8K:/db2$ps -ef |grep -i db2ara
      db2ara  9502850  3408174   0   Aug 10      -  1:58 db2acd 0
      db2ara 15597768  3408174   0   Aug 10      -  0:00 db2fmp (C)  0
      db2ara 57737264  3408174   0   Aug 10      -  0:00 db2fmp (idle) 0
      db2ara 59244648  3408174   0   Aug 10      - 15:40 db2sysc 0
      db2ara  8126806        1 120   Jun 14     - 66590:17
/db2/db2ara/sqllib/bin/db2bp 23658894A550 5 A
      db2ara 19988938 59244648   0   Aug 10      -  0:00 db2vend (PD Vendor
Process - 258)
      db2ara 50069994  3408174   0   Aug 14      -  0:00 db2fmp (idle)
```

2.1.4　实例的相关命令

在实例创建后，可以执行实例相关的命令来管理实例。在使用实例之前，必须更新每个用户的数据库环境，以便该环境可以访问实例并运行 DB2 实例相关命令。在运行这些命令之前，首先要配置好实例的运行环境，这适用于所有用户(包括管理用户)。而且在执行这些命令时一定要确保具有足够的权限。实例相关的命令对权限要求很高，例如 db2icrt 和 db2idrop 需要 root 权限才能执行。而除了这两个命令，其他实例命令需要具有 SYSADM、SYSCTRL 或 SYSMAINT 的权限才能运行(第 14 章中有关于权限的详细讲解)。

1. 配置实例的运行环境

我们都知道在 Linux/UNIX 环境中，在用户级上强制实施高安全策略时，与某个用户账户关联的文件和进程不能被其他用户直接访问。默认情况下，创建新的实例时，会在实例目录下生成特殊的 DB2 环境脚本 db2profile(Windows 下为 db2profile.bat)，每次实例所有者登录到系统时都要使用该文件配置其环境。这些脚本设置对数据库环境的访问，允许实例所有者执行 DB2 命令。为了让系统上的其他用户访问实例和 DB2 环境，他们也必须运行同样的脚本，否则将无法访问 DB2 实例运行环境。如下所示是由于没有设置 DB2 实例

运行环境而导致无法执行相关 DB2 命令的例子：

```
DS8K:/db2$db2 connect to sample
ksh: db2:  not found.
```

可以通过设置 DB2 实例所有者的.profile 文件(或是由.profile 文件引用的 db2profile 文件)来配置其他用户的 DB2 运行环境。如果使用 Bourne 或 Korn shell，那么可以编辑目标用户账户的.profile 文件，使目标用户登录到系统时自动运行 db2profile 脚本(创建实例时默认会加到实例用户的.profile 文件中)。对于 C shell 用户，可以编辑.login 文件来运行 db2cshrc 脚本文件。为了选择要使用的实例的用户，请在用户的.profile 或.login 脚本文件中添加下列语句，或者在用户需要访问 DB2 的终端窗口中执行下列语句(注意需要句点(.)和空格)：

```
Bourne 或 Korn shell: . INSTHOME/sqllib/db2profile ----注意前面的“.” INSTHOME
                                                      是实例目录
C shell: source INSTHOME/sqllib/db2cshrc
```

当成功创建实例后，检查实例目录下的.profile 是否调用 db2profile，如下所示：

```
# The following three lines have been added by IBM DB2 instance utilities.
if [ -f INSTHOME/sqllib/db2profile ]; then
    . INSTHOME/sqllib/db2profile
fi
```

在配置好 DB2 的运行环境后，下面讲解一些实例相关的命令。

列出实例

db2ilist 命令列出机器上的 DB2 实例：

```
DS8K:/db2$db2ilist
PROD
DEV
DB2
--系统中存在三个实例
```

迁移实例

```
db2imigr instanceName
        [/?]
        [/q]
        [/a:authType]
        [/p:instanceProf]
        [/u:username,password]
instanceName
```

```
        实例名
/?    此用法信息
/q    安静方式
/a    authType 是实例的认证类型(SERVER、CLIENT 或 SERVER_ENCRYPT)
/p    instanceProf 是迁移实例的实例概要文件路径
/u    username,password 用于指定 DB2 服务的账户名和密码。在迁移分区实例时，此选项是
      必需的
```

更新实例配置

如果通过安装“程序临时性修订(PTF)”或补丁更新了数据库管理器，那么应使用 db2iupdt 命令(需要 root 用户，在 DB2 V9.1 之后会在升级后自动调用此命令)来更新所有现有数据库实例。

db2iupdt 命令可在 DB2PATH\bin 目录中找到。要更新实例配置，请使用 db2iupdt 命令。按如下所示使用该命令：

```
db2iupdt InstName
      /u:username,password
      [/p:instance profile path]
      [/r:baseport,endport]
      [/h:hostname]
      [/?]
      [/q]
      [/a:authType]
InstName 实例名
/u       用于指定 DB2 服务的账户名和密码。当创建分区数据库实例时，此选项是必需的
/p       用于指定已更新实例的新的实例概要文件路径
/r       用于指定当在 MPP 方式下运行时，分区数据库实例要使用的一系列 TCP/IP 端口。如果
         指定此选项，那么本地机器的 services 文件将使用下列条目进行更新：
         DB2 InstName      baseport/tcp
         DB2 InstName END   endport/tcp
/h       用于覆盖默认 TCP/IP 主机名(假设当前机器有多个 TCP/IP 主机名)
/?       此用法信息
/q       安静方式
/a       authType 是实例的认证类型(SERVER、CLIENT 或 SERVER_ENCRYPT)
示例：如果在创建实例后安装了 DB2 工作组服务器版或 DB2 企业服务器版，可输入以下命令来更
新实例
    db2iupdt -u db2fenc1 db2inst1
```

注意：

db2imigr 和 db2iupdt 的区别是：db2iupdt 通常是小版本打补丁，而 db2imigr 通常是大版本迁移。例如从 DB2 V8 到 DB2 V9.1 用 db2imigr，而从 DB2 V9.5.0.4 到 V9.5.0.7 用 db2iupdt。在 DB2 V9.7 中，新增加了 db2iupgrade 命令，可以替代 db2imigr 命令的功能。

2. 自动启动实例

在 Windows 操作系统中，默认情况下，安装期间创建的数据库实例设置为自动启动，使用 db2icrt 创建的实例设置为手动启动。要更改启动类型，需要转至“服务”面板并在其中更改 DB2 服务的属性(手动或自动)。

在 UNIX 或 Linux 操作系统中，要允许实例在每次系统重新启动后自动启动，请输入以下命令：

```
db2iauto -on <instance name>--其中<instance name>是实例的登录名
```

在 UNIX 或 Linux 操作系统中，要阻止实例在每次系统重新启动后自动启动，请输入以下命令：

```
db2iauto -off <instance name>--其中<instance name>是实例的登录名
```

3. 启动实例

在正常业务操作期间，可能需要启动或停止 DB2 数据库。例如，必须启动一个实例，然后才能执行下列某些任务：连接至该实例中的数据库、预编译应用程序、将程序包绑定至数据库或访问主机数据库。

在Linux或UNIX系统中,启动实例之前需要以具有SYSADM、SYSCTRL或SYSMAINT权限的用户标识或名称进行登录；或者作为实例所有者登录。在 Windows 中启动实例，用户账户必须具有 Windows 操作系统定义的、用于启动 Windows 服务的正确特权。用户账户可以是 Administrators、Server Operators 或 Power Users 组的某个成员。启用了扩展安全性之后，默认情况下，只有 DB2ADMNS 和 Administrators 组的成员才能启动数据库。

要使用命令行启动实例，请输入：

```
db2start
```

在 Windows 中，db2start 命令将 DB2 数据库实例作为 Windows 服务来启动。通过在调用 db2start 时指定“/D”开关，仍然可以在 Windows 中将 DB2 数据库实例作为进程运行。还可使用“控制面板”或 NET START 命令将 DB2 数据库实例作为服务启动。

4. 连接至实例和从实例断开

在所有平台上，要与另一个可能是远程的数据库管理器的实例连接，请使用 ATTACH 命令。要从实例断开，请使用 DETACH 命令。

要使用命令行与实例连接，请输入：

```
db2 attach to <instance name>
```

例如，要连接至节点目录中先前编目的称为 testdb2 的实例：

```
db2 attach to testdb2
```

再如，在对 testdb2 实例执行维护活动后，要使用命令行从实例断开，请输入：

```
db2 detach
```

或者输入：

```
db2 terminate
```

注意：

这两个命令在实际生产中很少用到，因为在 Linux/UNIX 环境中，每次都会通过某个用户登录操作系统，这已经隐含地连接了实例。

5. 停止实例

我们有时可能需要停止数据库管理器的当前实例。要在 Linux 或 UNIX 系统中停止实例，必须以具有 SYSADM、SYSCTRL 或 SYSMAINT 权限的用户标识或名称登录或连接至实例；或者作为实例所有者登录。在 Windows 中，停止实例的用户账户必须具有 Windows 操作系统定义的正确特权。用户账户可以是 Administrators、Server Operators 或 Power Users 组的某个成员。在停止实例之前，要停止与数据库连接的所有应用程序和用户，要确保没有关键性的或极重要的应用程序在运行。

要使用命令行来停止实例，请输入：

```
db2stop
```

如果停止期间仍然有应用连接，这时会出现“数据库管理器未停止，因为数据库仍在活动”的错误：

```
    DS8K:/db2$db2stop
    2012-08-24 10:49:19     0    0   SQL1025N  The database manager was not
stopped because databases are still active.
    SQL1025N  The database manager was not stopped because databases are still
active.
```

如果要强制所有应用程序和用户与数据库断开，那么需要输入如下命令：

```
db2stop force
```

这时，所有连接上数据库且未提交的应用将强制回滚。

在 Windows 中，除了使用命令行之外，还可以通过“控制面板-服务”或 NET STOP 命令停止实例。

2.1.5 DB2INSTANCE 变量介绍

如果系统中有多个实例，那么如何在各个实例之间进行切换以及如何同时启动多个实例呢？这就需要使用 DB2INSTANCE 环境变量。环境变量是操作系统层面的，是在操作系统级别上定义的变量。最常使用的 DB2 环境变量是 DB2INSTANCE，该环境变量允许指定当前活动实例，所有命令都将应用于该实例。例如，如果将 DB2INSTANCE 设置成“PROD”，那么发出命令“create database mydb”会创建出与实例“PROD”相关的数据库。但是如果想创建与实例“DB2”相关的数据库，那么首先必须将 DB2INSTANCE 变量的值更改成“DB2”。

可以使用控制面板(在 Windows 中)/db2profile(在 UNIX 中)来设置环境变量的值，从而保证下次打开窗口/会话时该值不变。但是，如果想在给定的窗口/会话中临时更改该值，那么在 Windows 中可以使用操作系统的“set”命令，在 UNIX 中可以使用“export”或“setenv”命令。例如，在 Windows 平台上，下面这个命令：

```
set DB2INSTANCE=DB2
```

会将环境变量 DB2INSTANCE 的值设置成“DB2”。**使用 set 命令时常犯的错误是在等号(=)前后留有空格。绝对不能有空格！**

要查看该变量的当前设置，可以使用下面三个方法中的任何一个：

```
echo %DB2INSTANCE% (Windows only);echo $DB2INSTANCE(Linux/UNIX)
set DB2INSTANCE
db2 get instance
```

示例：假设系统中有多个实例，下面举例说明如何通过设置 DB2INSTANCE 环境变量来启动多个实例。

```
C:\Program Files\IBM\SQLLIB\BIN>db2ilist
PROD
DEV
DB2
--系统中存在三个实例
C:\Program Files\IBM\SQLLIB\BIN>set DB2INSTANCE
DB2INSTANCE=DB2
```

```
--当前活动实例是 DB2 实例
C:\Program Files\IBM\SQLLIB\BIN>set DB2INSTANCE=PROD
C:\Program Files\IBM\SQLLIB\BIN>set DB2INSTANCE
DB2INSTANCE=PROD
C:\Program Files\IBM\SQLLIB\BIN>db2 get instance
--当前数据库管理器实例是 PROD
--db2 get instance 命令用来判断当前在哪个实例下
C:\Program Files\IBM\SQLLIB\BIN>db2start
--这时启动的是 prod 实例
```

其实，DB2INSTANCE 环境变量类似 Oracle 中的变量 ORACLE_SID、Informix 中的变量 INFORMIXSERVER，主要用于在多个实例间进行切换。这个变量在 Windows 中很常用，在 Linux/UNIX 中由于每个实例都有同名的用户，因此当使用这个用户登录时已经隐含地连接了这个实例。所以相对来说在 Linux/UNIX 中很少用到这个变量。

2.1.6　删除实例

要删除实例，必须具有 root 权限，在 Windows 中必须具有系统管理员权限。删除实例之前，确保所有的应用已经断开实例并且实例已经停止。

要使用命令行删除实例，请输入：

```
db2idrop <instance_name>
```

db2idrop 命令从实例列表中删除实例条目，并删除实例所有者 home 目录下的 sqllib 子目录。所以删除实例时千万要小心，如有必要请在删除实例之前备份实例目录。

注意：

在 Linux/UNIX 操作系统中，试图使用 db2idrop 命令删除实例时，会生成一条消息，说明不能删除 sqllib 子目录，并且正在 adm 子目录中生成几个具有.nfs 扩展名的文件。adm 子目录是安装了 NFS 的系统，而这些文件在服务器上是受控的。必须从安装目录的文件服务器中删除*.nfs 文件，然后方可删除 sqllib 子目录。

2.1.7　配置实例

每个实例在创建后，都有实例配置文件(db2systm)，实例配置文件控制实例的安全、通信、管理和资源的分配。可以根据需要查看、更改和复位这个配置参数。这个配置文件是二进制的，只能通过命令来修改。

可使用 db2 get dbm cfg 命令来查看当前实例的配置参数。

要查看当前实例配置参数的当前值，请输入：

```
db2 get dbm cfg
```

这会显示在安装该产品期间指定为默认配置参数的当前值，或在先前更新配置参数期间指定的那些值。

可在命令行使用 update dbm config 更新实例配置文件。要更新实例配置文件中的个别条目，请输入：

```
db2 update dbm cfg using ..
```

要将配置参数复位为建议的默认值，请输入：

```
db2 reset dbm cfg
```

在某些情况下，对实例配置文件的更改仅在将更改装入内存后才生效(在执行 db2stop 之后，执行 db2start 时生效)。

关于实例配置文件，已超出了本章的讨论范围，我们会在后面的章节中为大家详细讲解如何合理地设置实例设置参数以使实例稳定、安全、高效地运行。

2.2 管理服务器

2.2.1 管理服务器的概念

DAS 是数据库管理服务器(Database Administration Server)的缩写，是驻留在数据库服务器机器上的特殊实例。DAS 是特殊的 DB2 管理控制点，用于帮助其他 DB2 服务器执行远程管理任务(其他远程 DB2 服务器需要安装 Database Administration Client，DB2 V9 以后更名为 DB2 Client)。DB2 数据库管理服务器(DAS)响应来自远程 DB2 服务器管理工具和配置助手(CA)的请求。如果要使用“客户机配置助手”来发现远程数据库或随 DB2 产品一起提供的图形工具(例如控制中心或任务中心)，那么 DAS 必须正在运行。

DAS 与实例是一对多的关系，即 DB2 数据库管理服务器中只能有一个 DAS，但这个 DAS 可以同时管理多个实例，如图 2-5 所示。

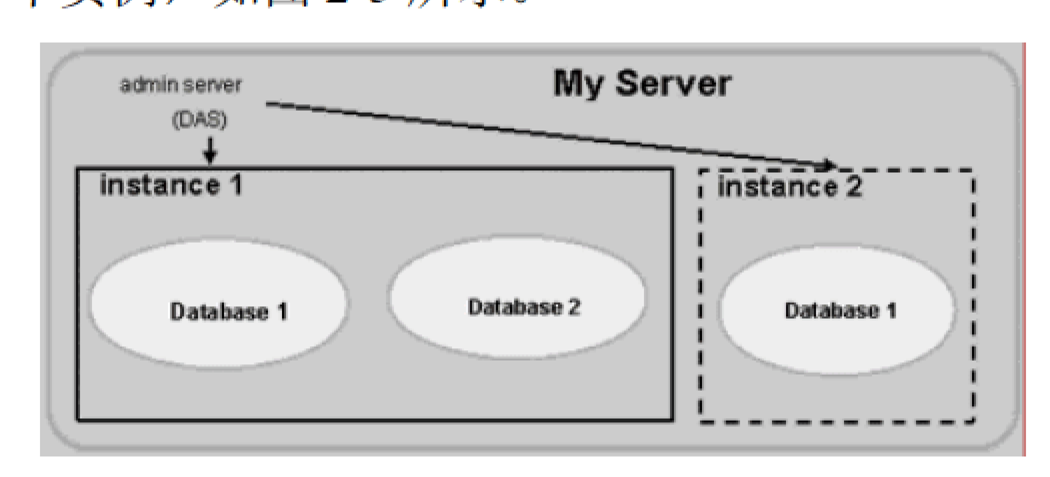


图 2-5　DAS 与实例的关系

DB2 全局注册变量参数 DB2ADMINSERVER 定义了管理 DB2 数据库服务器的 DAS：

```
E:\IBM\SQLLIB\BIN>db2set -all |find /i "das"
[g] DB2ADMINSERVER=DB2DAS00
```

其中，DAS 允许使用 DB2 工具(例如 Control Center、Client Configuration Assistant 和任务中心)对服务器上的数据库进行本地和远程管理。事实上，为了利用这些工具，DAS 实例必须存在并被启动。DB2 图形管理工具与 DAS 共同使诸如以下的任务得以被执行：

- 从远程客户机上管理、调度 DB2 或操作系统脚本以在服务器上运行
- 使用 DB2 Discovery 自动设置客户机/服务器连接
- 启动或停止远程 DB2 实例

举个例子，假设安装了 DB2 的数据库服务器远在北京机房，而现在想在上海利用 DB2 提供的图形化管理工具来管理这个北京机房的 DB2，那么必须在北京的 DB2 服务器上配置启动管理服务器。其实 DAS 不是必须创建的，假设不需要远程地使用图形化管理界面的工具来管理 DB2，那么可以不创建 DAS，通过命令行远程地管理 DB2 不需要 DAS 存在。

2.2.2　创建管理服务器

管理服务器必须位于将要被管理和搜索的每台服务器上。每个数据库服务器机器上只能创建一个 DAS 实例。除非另外指定，否则这个实例会在 DB2 安装过程中自动建立。默认情况下，DAS 实例名在 Windows 中是 DB2DAS00，在 UNIX /Linux 中默认是 DB2AS。如果查看一下在 Windows 机器上运行的服务，将看到列出了 DB2DAS00 服务，并且被配置成自动启动，如图 2-6 所示。



图 2-6　在 Windows 中默认创建的 DAS

如果在 DB2 安装过程中未自动创建 DAS 实例，那么必须手动创建。在 UNIX/Linux 中，创建命令是 dasicrt name。在 Windows 中，创建命令是 db2admin create。

1. 在 Windows 中创建 DAS

在 Windows 中，创建 DAS 时必须具有本地 Administrator 权限。如果期望使用特定用户账户，那么在发出 db2admin create 时必须使用“/USER:”和“/PASSWORD:”。

在数据库服务器中只能有一个 DAS。如果已经创建了 DAS，那么需要通过发出

db2admin drop 将之删除。

创建 DAS 后，可能需要更改 Windows 中运行 DAS 服务的用户标识。可使用 db2admin 命令设置或更改登录账户，如下所示：

```
db2admin setid <username> <password>
```

其中，<username>和<password>是具有本地管理员权限的账户的用户名和密码。在运行此命令前，必须使用具有本地管理员权限的账户或用户标识登录计算机。

注意：

在 Windows 中，不应使用控制面板中的服务实用程序来更改 DAS 的登录账户，因为这样将不会为登录账户设置某些必需的访问权限。始终使用 db2admin 命令来设置或更改 DB2 管理服务器(DAS)的登录账户。

当安装过程与 DAS 相关时，作为在安装过程期间发生的情况的概述，考虑下列事项：在 Windows 平台上，使用具有正确的服务创建权限的账户登录想要创建 DAS 的计算机。

当创建 DAS 时，可以选择是否提供账户名和密码。如果账户名和密码有效，它们将标识 DAS 的所有者。不要使用为 DAS 创建的用户标识或账户名作为“用户账户”。将账户密码设置为“密码永不到期”。在创建 DAS 之后，就可以使用 db2admin setid 命令提供账户名和密码，建立或修改 DAS 的所有权。

2. 在 Linux 和 UNIX 中创建 DAS

在 Linux 和 UNIX 中，要想创建 DAS，必须首先创建和 DAS 一样的用户及用户所属的组。之所以需要创建用户，主要是因为需要以用户的 home 目录作为 DAS 实例目录，存放 DAS 相关的实例目录结构。

使用 dascrt 创建实例的步骤如下：

(1) 使用 root 权限登录(在 Windows 中使用系统管理员账户登录)。

(2) 首先利用操作系统命令(mkuser、useradd；mkgroup、groupadd)创建 DAS 的用户和组并设置密码，一般来说需要创建表 2-2 所示的用户和组。表 2-2 中的用户和组是使用 DB2 安装向导期间由 DB2 默认生成的。实际生产中，可以根据需要创建自己特定的组名和用户名，此处仅为说明作用。

表 2-2　默认生成的用户和组

用　　户	示例用户名	示 例 组 名
DB2 管理服务器用户	dasusr1	dasadm1

(3) 运行 dascrt 命令。例如，在 Linux 或 UNIX 操作系统中：

```
DB2DIR/instance/dascrt -u DASuser
```

其中，DB2DIR 是 DB2 安装目录。在 Linux/UNIX 操作系统中，默认的 DB2 安装目录是/opt/IBM/db2/V9.7。

成功执行 dascrt 后，运行 daslist 命令，可以查看已创建的 DAS：

```
DB2DIR/instance/daslist
DASuser
```

2.2.3 管理服务器的相关命令

启动和停止 DB2 管理服务器

要在 Windows 中手动启动或停止 DAS，必须首先使用属于 Administrators、Server Operators 或 Power Users 组的账户或用户标识登录至计算机。要在 Linux/UNIX 中手动启动或停止 DAS，账户或用户标识必须成为 *dasadm_group* 的一部分。*dasadm_group* 在 DAS 配置参数中指定。要启动或停止 DAS，可以使用 db2admin start 或 db2admin stop 命令。

列示 DB2 管理服务器

使用 db2admin 命令可列示计算机中 DAS 的名称。

更新 DB2 管理服务器(Linux/UNIX)

在 Linux/UNIX 操作系统中，可以在一台机器上多次安装 DB2 数据库产品。虽然 DB2 管理服务器只能有一个，但是所有 DB2 副本都将使用它。DAS 可以从任何一个 DB2 副本创建，并将在与该 DB2 副本相同的修订包级别运行。如果对该 DB2 副本打了补丁，那么应发出 dasupdt 命令来更新 DAS。

更新 DAS 的步骤如下：

(1) 使用超级用户权限(通常作为“root”用户)登录到计算机。

(2) 发出 dasupdt 命令：

```
DB2DIR/instance/dasupdt
```

其中，DB2DIR 表示安装 DB2 副本的位置。一旦完成此命令，DAS 就将在 DB2 副本的新修订包级别运行。

2.2.4 删除管理服务器

使用 db2admin 或 dasdrop 命令可以从 Windows 或 Linux/UNIX 平台删除 DAS。

要删除 DAS，需要在 Windows 系统中确保有正确权限登录。在 Linux/UNIX 操作系统中，首先作为具有 DASADM 权限的用户登录，然后使用 db2admin stop 来停止 DAS。最后，作为 root 用户登录，并使用 dasdrop 命令删除 DAS，如下所示：

```
dasdrop
```

注意：

dasdrop 命令会删除 DB2 管理服务器主目录下的 das 子目录。

2.2.5 配置管理服务器

可使用 DB2 get admin cfg 命令来查看 DAS 的当前配置参数。

要查看与 DAS 相关的 DB2 管理服务器配置参数的当前值，请输入：

```
db2 get admin cfg
```

这会显示在安装产品期间指定为默认配置参数的当前值，或在先前更新配置参数期间指定的那些值。

可在命令行使用 update admin config 更新 DAS 配置文件，必须从与 DAS 具有相同安装级别的实例使用命令行。要更新 DAS 配置文件中的个别条目，请输入：

```
DB2 update admin cfg using
```

要将配置参数复位为建议的默认值，请输入：

```
db2 reset admin cfg
```

在某些情况下，对 DAS 配置文件的更改仅在将更改装入内存后才生效(在执行 db2admin stop 之后，执行 db2admin start 时生效；对于 Windows 平台，在停止并启动该服务时生效)。在其他情况下，配置参数是可联机配置的(不必重新启动 DAS 就可以使更改生效)。

关于 DAS 这个概念，主要是希望大家了解 DB2 数据库中有这个概念，它主要执行远程图形化界面管理功能，这个概念在 DB2 中相对来说不是特别重要。在很多 UNIX/Linux 环境中，客户都是通过命令行来进行远程管理的，这种情况下可以不创建 DAS，而在 UNIX/Linux 生产环境中，建议删除 DAS，通过命令行方式操作 DB2。

注意：

从 DB2 V9.7 开始，不推荐使用控制中心工具和 DAS，在将来的 Discover 版本中可能会将之删除。可以考虑用新的 GUI 工具套件来替代控制中心工具，详情可参考 IBM Data Studio 和 Optim 提供的解决方案。



创建数据库和表空间

在创建完实例后，下一步要做的工作就是创建数据库和存放数据库对象的容器——表空间。

本章主要讲解如下内容：

- 创建数据库
- 设计表空间
- 缓冲池

3.1 创建数据库

DB2 数据库的概念

在 DB2 中，一个 DB2 实例可以同时管理多个 DB2 数据库，而一个 DB2 数据库只能由一个 DB2 实例管理，DB2 数据库与 DB2 实例是一种松散耦合的关系。在 UNIX 或 Linux 中，创建数据库所生成文件所属的用户和组都是 DB2 实例的所有者，即创建实例的用户和组。实例和数据库的关系如图 3-1 所示。

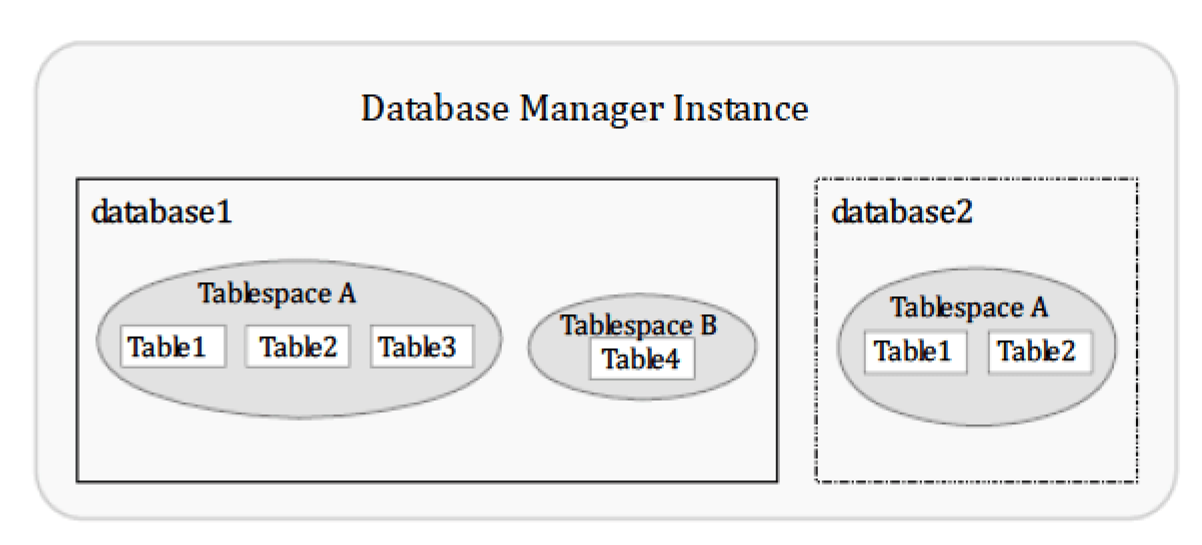


图 3-1　实例和数据库的关系

DB2 数据库实际上由对象集合组成。从用户的角度来看，数据库是一组通常以某种方式相关联的表。从数据库管理员的角度来看，数据库比这要复杂一些。实际的数据库包含许多逻辑对象和物理对象：

- 表、视图、索引、模式、触发器、存储过程、程序包等数据库对象
- 缓冲池、日志文件、表空间
- 物理存储、表空间容器、目录、文件系统或裸设备

这些对象中的一部分(比如表或视图)帮助决定如何对数据进行组织；其他对象(比如表空间)涉及数据库的物理实现；最后，一些对象(比如缓冲池和其他内存对象)处理如何管理数据库性能；另外一些对象(比如日志文件)处理数据库的可恢复性。数据库的逻辑结构如图 3-2 所示。

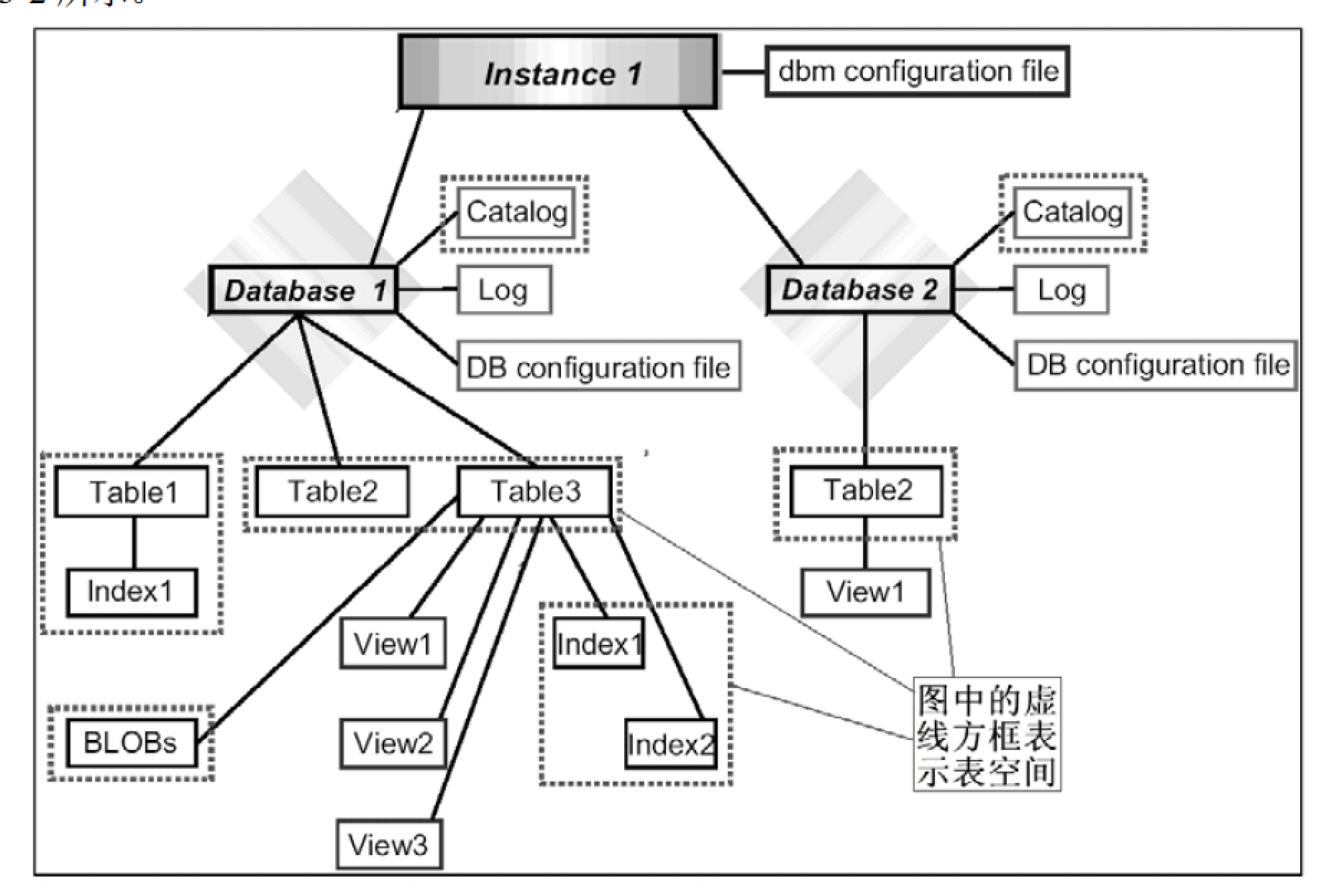


图 3-2　数据库的逻辑结构

DBA 应该首先关注数据库的物理设计，而不是直接研究所有可能的参数和对象组合。而数据库的逻辑设计，比如表的字段属性设计，则主要由应用设计人员完成。DBA 的核心工作之一就是研究如何创建数据库并分配数据库所需的磁盘存储。要正确地解决这个问题，需要了解数据库中的基本对象以及它们如何映射到物理磁盘存储。

3.1.1　DB2 数据库存储模型

DB2 利用逻辑存储模型和物理存储模型来处理数据。用户操作的实际数据存放在表中。表由行和列组成，用户并不知道数据的物理表示。这一事实有时候称为数据的物理独立性。

表本身存放在表空间中，表空间是存放表的储藏室(容器)，一个表空间可以包含多个表。同时，表空间在物理上又对应着若干个表空间容器。容器可以由目录名、裸设备名或文件名标识。容器被分配给表空间。表空间可以跨许多容器，这意味着可以突破操作系统对于一个容器可以包含的数据量的限制。这样一来，表空间就作为逻辑设计中的表与物理设计中的容器之间的桥梁，表通过表空间实实在在地将数据存放到容器(文件或目录)中。图 3-3 说明了所有这些对象之间的关系。

从图 3-3 中可以看到：数据库中有很多表空间，可以把数据库看作很多个表空间的集合；可以根据需要创建多个表空间。而用户创建的表、索引等数据库对象存放在表空间中。表直接面向应用。而同时表空间又和底层的物理存储对应，表空间可以有多个容器，而容器处在底层存储之上。所以通过表空间数据库实现了物理存储和逻辑存储的统一。表空间是 DB2 中最重要的概念之一。

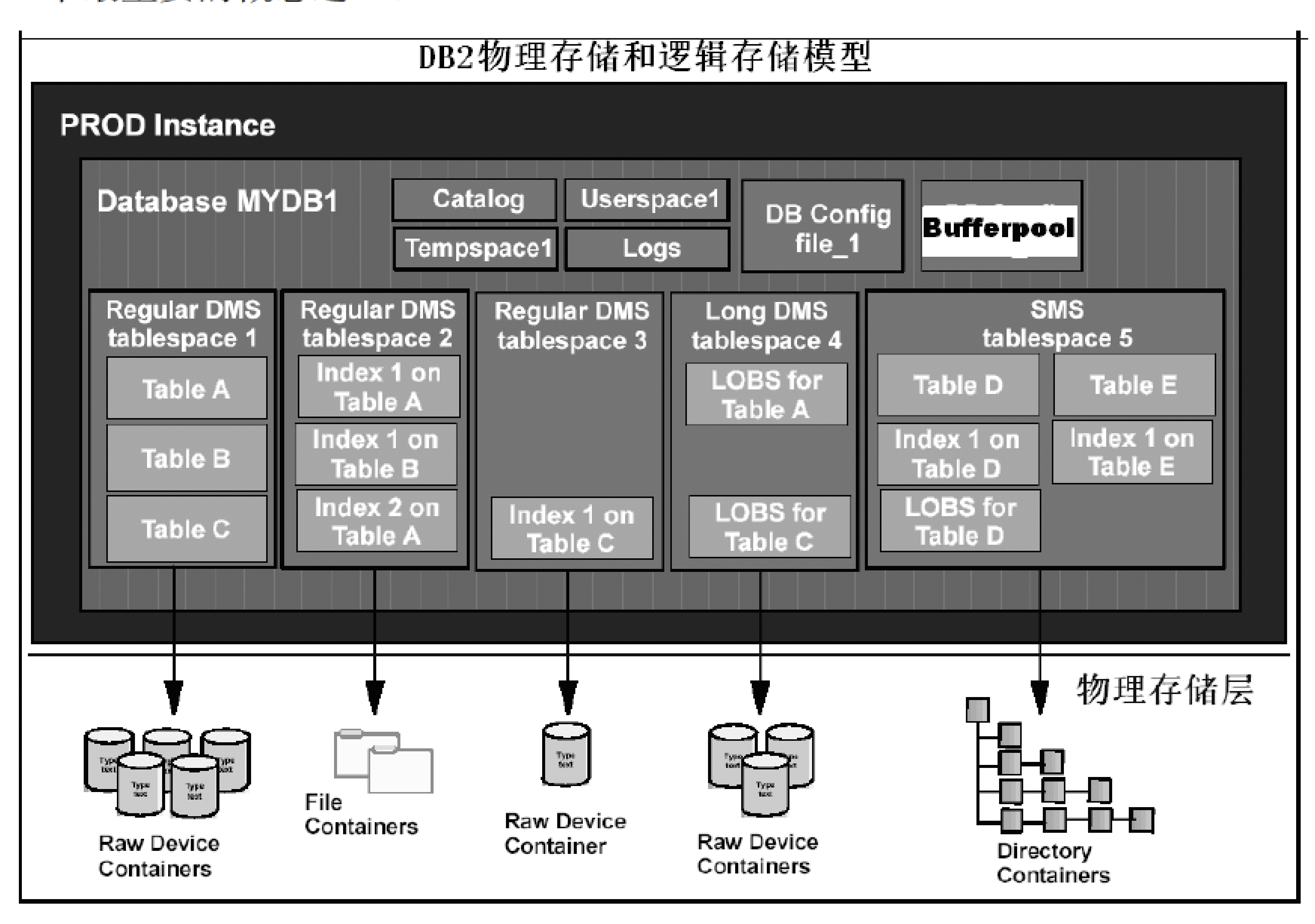


图 3-3　数据库、表空间、表和表空间容器之间的关系

下面首先讲解数据库的物理存储。我们都知道操作系统的最小存储单位是块(block)，在 Linux 和 UNIX 中，最小的块是 512 字节；在 Windows 中，最小的存储单位为 1KB。而数据库中最小的存储单位是数据页(datapage)，它是 DB2 读写的最小单位。DB2 数据库中有 4KB、8KB、16KB 和 32KB 几种数据页。可以根据业务类型(OLAP、OLTP 等)和表的大小来选择合适的数据页。DB2 数据库在写物理存储时，为了保证写的吞吐量，引入了更大的单位 extent，它是整数倍的 datapage 的大小。这个可以在创建表空间时指定 extentsize 大小来确定。而表空间容器又是由很多个 extent 组成，同时表空间又由很多容器组成，它们之间的关系如图 3-4 所示。

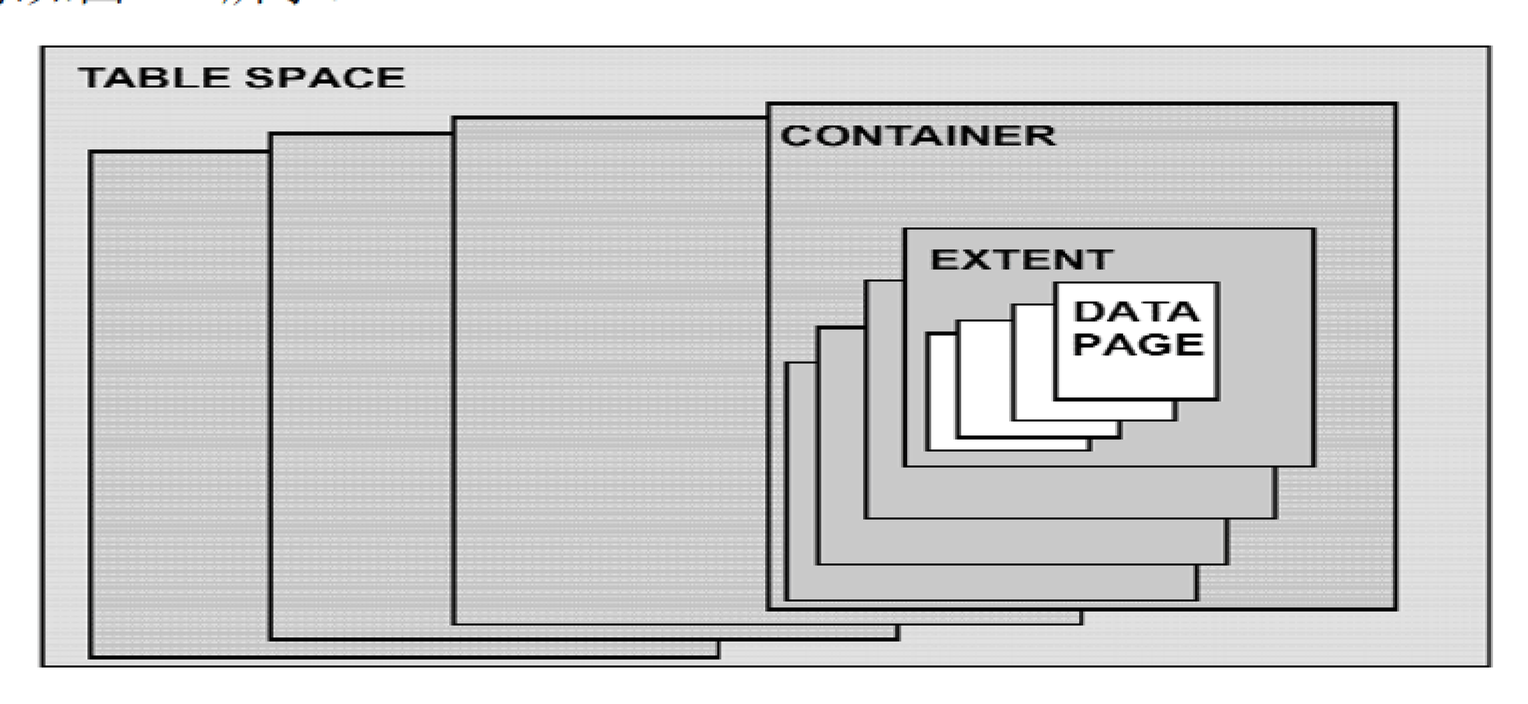


图 3-4　表空间、容器、extent 和数据页之间的关系

从上面的图 3-4 中可以得知，创建的表最终存储在底层的表空间容器中，那么 DB2 如何来写容器呢？请看下面的图 3-5。

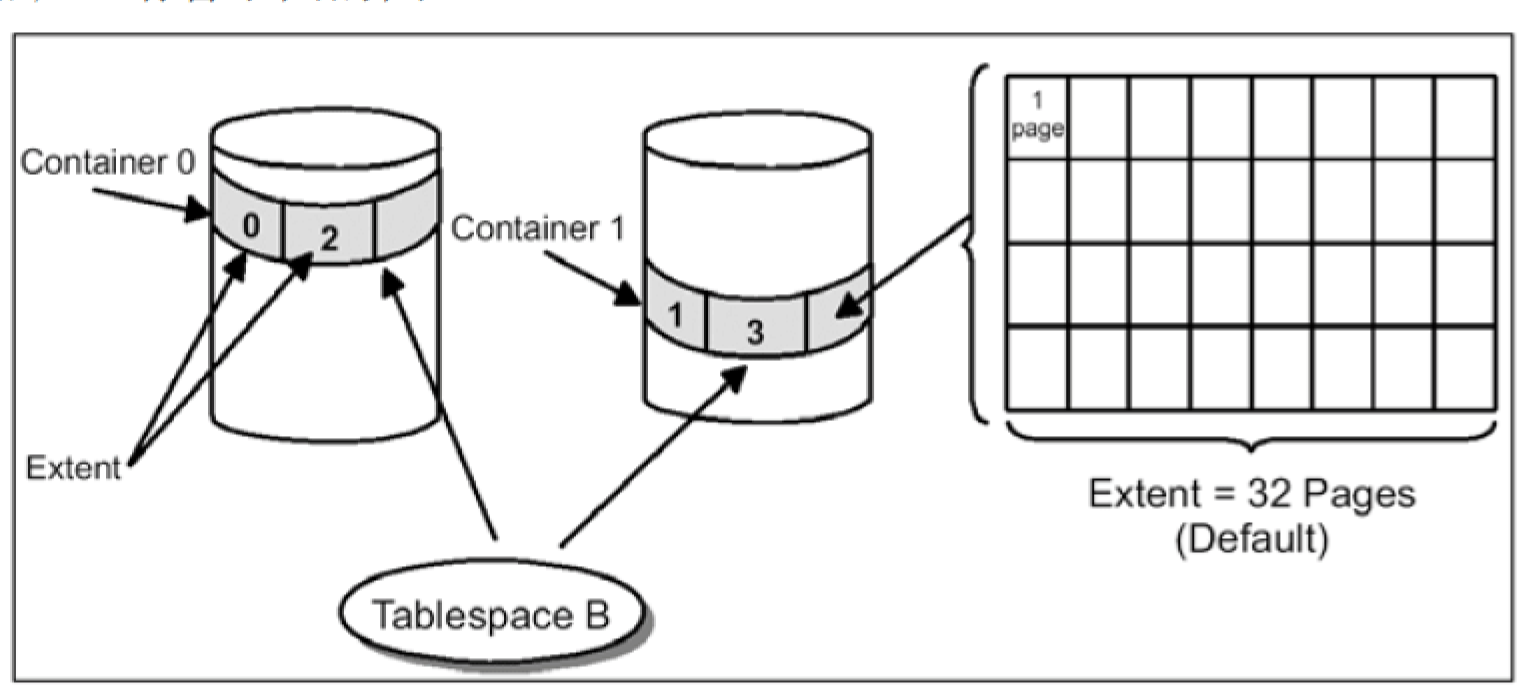


图 3-5　DB2 写容器的过程

我们知道表空间是由很多容器组成的，每次在写容器的时候，写的单位为扩展数据块 extent。extent 的大小可以在创建数据库和表空间的时候通过 extentsize 大小指定；而 extent

又是由很多 extentsize(默认为 32)个数据页(datapage)组成。datapage 是 DB2 数据库中最小的存储单位，是每次读写的最小 I/O 单位。

图 3-6 显示了具有两个 4KB 页扩展数据块(extent)大小的 HUMANRES 表空间，它有 4 个容器，每个容器有少量已分配的扩展数据块(extent)。DEPARTMENT 和 EMPLOYEE 表各有 7 页，并且跨所有 4 个容器。一个 extent 同时只能被一个表写，也就是说，不可能两张表同时共用一个 extent，因为写容器的最小单位是 extent，当往表中插入数据，但却发现没有空间时，DB2 就会为表分配 extent。

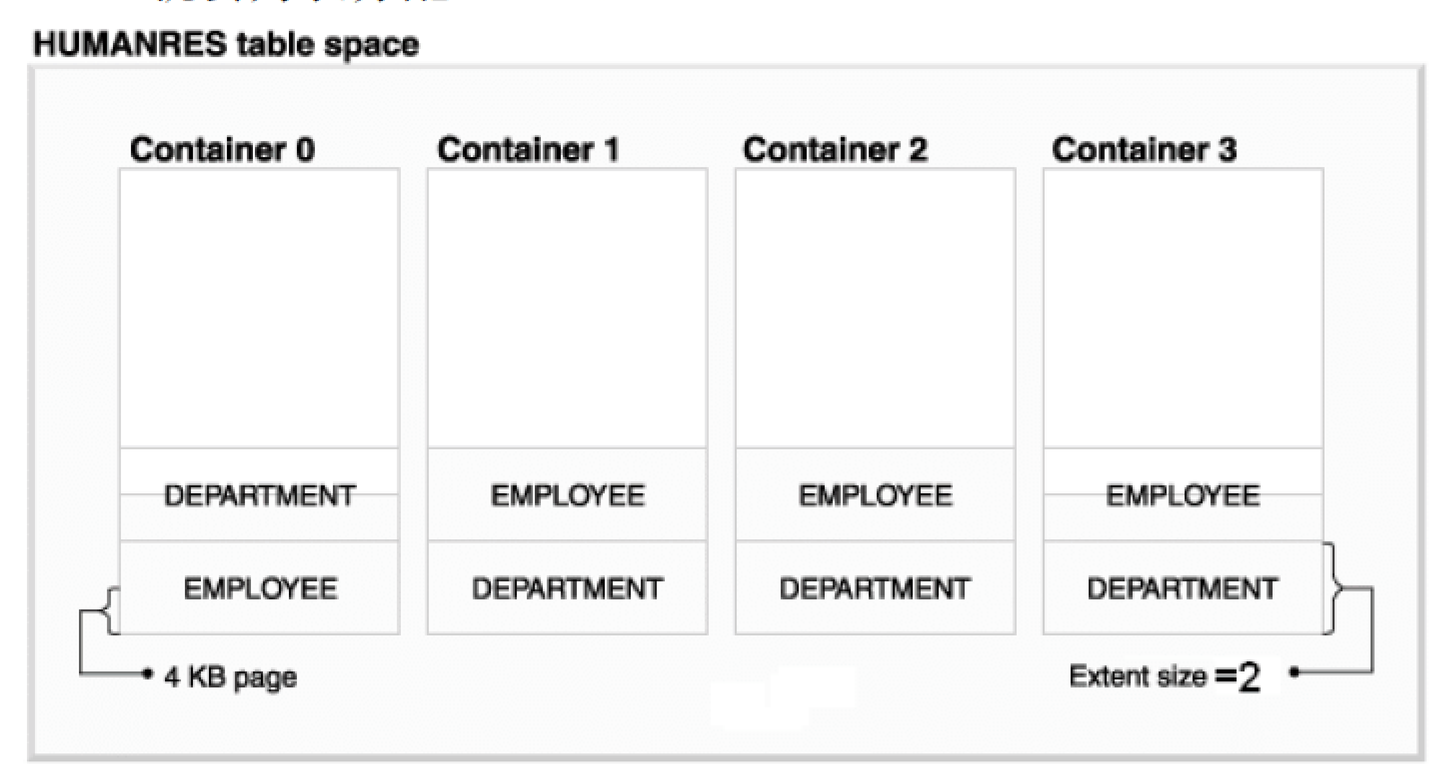


图 3-6　表空间中的容器、扩展数据块(extent)、数据页(datapage)和表空间之间的关系

通过上面的讲解，我们发现在数据库的物理存储和逻辑存储模型中，表空间连接了物理存储模型和逻辑存储模型，扮演承上启下的角色。在逻辑上，它向上面对的是数据库，向下它是存放表的容器，面向的是表；而同时表空间又在物理上映射底层的表空间容器——物理存储。表空间是数据库性能调优的重点，而数据库创建工作的绝大部分都是围绕着表空间进行的，下面首先讲解数据库中表空间这个最重要的概念。

我们首先了解表空间的类型，DB2 中有 3 种管理类型的表空间。

3.1.2　表空间管理类型

DB2 支持 3 种管理类型的表空间：

- 系统管理的空间(System-Managed Space，SMS)：在这里，由操作系统的文件系统管理器分配和管理空间。在 DB2 V9 之前，如果不带任何参数创建数据库或表空间，就会导致所有表空间作为 SMS 对象创建。这种表空间依赖底层的操作系统(例如 AIX、HP-UX 或 Windows)来进行空间管理。

- 数据库管理的空间(Database-Managed Space，DMS)：在这里，由 DB2 数据库管理程序控制存储空间。表空间容器可使用文件系统或裸设备。
- DMS 的自动存储(Automatic Storage With DMS)：自动存储实际上不是一种单独的表空间类型，而是一种处理 DMS 存储的不同方式。因为数据库管理的空间需要比较多的维护(见后面的 3.2.2 节)，在 DB2 V8.2.2 中引入了 DMS 自动存储以简化表空间管理。

SMS 表空间需要的维护非常少。但是，与 DMS 表空间相比，SMS 表空间提供的优化选项少且性能不好。

那么，应该选择哪种表空间设计呢？下面我们来看看这几种表空间的比较。

1. DMS、SMS 与 DMS 自动存储的比较

尽管下面的表 3-1 并不全面，但却包含在 DMS、DMS 自动存储和 SMS 表空间之间进行选择时要考虑的一些因素。

表 3-1　DMS、DMS 自动存储和 SMS 表空间的比较

特　性	SMS	DMS	DMS 自动存储
是否条带化(Striping)	是	是	是
默认类型	Version 8	无	Version 9
对象管理	操作系统	DB2	DB2
空间分配	按需增长/收缩	预先分配；大小可以收缩和增长，但是需要 DBA 干预	预先分配；可以自动增长
管理的简便性	最好；很少需要调优，甚至不需要	好，但是需要一些调优(例如 EXTENTSIZE PREFETCHSIZE)	最好；很少需要调优，甚至不需要
性能	不太好	很好；可通过利用裸设备多获得 5%到 10%的收益	最好；但是，不可以使用裸设备

SMS 表空间可以简化管理，DMS 表空间可以提高性能，除此之外，这两种存储模型之间最显著的差异是表空间的最大大小。在使用 SMS 表空间时，DBA 最多只能在表空间中存放 64GB(4KB 页大小)的数据。将页面大小改为 32KB，可以将这个限制扩大到 512GB，但代价是每个页面上的可用空间可能会更少。在 DB2 V9.7 里，DMS 模型会将大型表空间的限制扩大到 8TB(4KB 页面大小的情况下)。如果将页面大小改为 32KB，可用空间可以增长到 64TB。尽管还有让表大小突破 64GB 限制的其他方法，但是最简单的方法可能是一开

始就使用 DMS 的大型表空间。

2. DMS 与 DMS 自动存储

DB2 V8.2.2 引入了 DMS 自动存储的概念。DMS 自动存储允许 DBA 为数据库设置在创建所有表空间容器时可以使用的存储路径。DBA 不必显式地定义表空间的位置和大小，系统将自动地分配表空间。在 DB2 V9 中，数据库在创建时默认将启用 DMS 自动存储，除非 DBA 显式地覆盖这个设置。

启用 DMS 自动存储的数据库有一个或多个相关联的存储路径。表空间可以定义为“由自动存储进行管理”，容器由 DB2 根据这些存储路径进行分配。数据库只能在创建时启用自动存储。在 DB2 V9.7 之前，对于在最初没有启用自动存储的数据库，不能在以后启用这个特性，但是在 DB2 V9.7 及之后的版本中，未启用自动存储的现有数据库现在可转换为使用自动存储，同样，现有的 DMS 表空间现在可转换为使用自动存储，可使用 ALTER DATABASE 语句来为现有数据库启用自动存储。

下面的表 3-2 总结了管理 DMS 非自动存储和 DMS 自动存储之间的一些差异。

表 3-2　管理 DMS 非自动存储和 DMS 自动存储之间的差异

特　　性	非自动存储	自 动 存 储
容器的创建	必须在创建表空间时显式地提供容器	不能在创建表空间时提供容器；它们将由 DB2 自动地分配
容器大小的调整	在默认情况下，表空间大小的自动调整是关闭的(AUTORESIZE NO)	在默认情况下，表空间大小的自动调整是打开的(AUTORESIZE YES)
初始大小	不能使用 INITIALSIZE 子句指定表空间的初始大小	使用 INITIALSIZE 子句指定表空间的初始大小
容器的修改	可以使用 ALTER TABLESPACE 语句(ADD、DROP、BEGIN NEW STRIPE SET 等等)执行容器修改操作	不能执行容器修改操作，因为由 DB2 控制空间管理。但是可以通过 ALTER TABLESPACE (REDUCE)命令降低高水位和表空间大小
管理的简便性	可以使用重定向的恢复操作重新定义与表空间相关联的容器	不能使用重定向的恢复操作重新定义与表空间相关联的容器，因为由 DB2 控制空间管理

引入自动存储模型的主要目的是简化 DMS 表空间的管理，同时保持其性能特征。有的时候，DBA 必须定义使用的表空间的所有特征，但是许多应用程序都会从自动存储提供的简化管理获益。

3. DB2 存储模型小结

经过上面的讲解，我们对数据库、表空间、容器和数据库对象做个总结：

- 数据库是对象集合，包括表、索引、视图、大对象和触发器等。
- 这些对象存储在表空间中，表空间由表空间容器组成。
- 表空间可以由操作系统管理(SMS)，也可以由 DB2 管理(DMS、自动存储)。
- 表空间容器可以选择使用底层存储——文件系统、裸设备或操作系统目录。
- 表空间由很多扩展数据块(extent)组成，而 extent 又由 extentsize(可自己定义)个数据页(datapage)组成，数据页是最小的存储单位。
- 应该主要根据性能和维护因素决定要使用的表空间类型、扩展数据块的大小、数据页的大小和容器类型。

既然已经熟悉了不同类型的表空间，就该创建第一个数据库了。下面讲解如何创建数据库。

3.1.3 创建数据库

创建数据库有很多方法，可以选择在安装后打开“DB2 第一步”启动面板来创建数据库，这个已经在第 1 章讲过了。除此之外，还可以通过 CREATE DATABASE 命令和数据库创建向导来创建数据库，下面分别讲解如何使用这两种方法创建数据库。

1. 使用命令创建数据库

从命令行创建 DB2 数据库是相当简单的。要创建数据库，必须调用 DB2 命令行处理程序(Command Line Processor，CLP)。调用方法是在 DB2 程序组的 Command Line Tools 文件夹中选择 Command Line Processor，或者从操作系统命令行执行命令 db2cmd db2。

创建 DB2 数据库的语法如下：

```
CREATE DATABASE MYDB
```

你可能会问“就这么简单？”，是的，就这么简单！CREATE DATABASE 语句中唯一必需的选项就是数据库的名称。数据库的命名规则是：

- 数据库的名称可以由以下字符组成：a-z、A-Z、0-9、@、#和$。
- 名称中的第一个字符必须是字母表字符、@、#或$；不能是数字或字母序列 SYS、DBM 或 IBM。注意，数据库的名称不能超过 8 个字符。
- 数据库名称或数据库别名是唯一的字符串，包含前面描述的 1 到 8 个字母、数字或键盘字符。

当然，有很多选项可供使用；不必只输入名称。下面研究一下这个命令实际上会导致什么情况。

1) DB2 创建了什么

在发出 CREATE DATABASE 命令时，DB2 会创建许多文件。这些文件包括日志文件、配置信息、历史文件和 3 个默认的表空间。这些表空间是：

- SYSCATSPACE：这是保存 DB2 系统编目的地方，系统编目跟踪与 DB2 对象相关联的所有元数据，即通常所说的“数据字典”。
- TEMPSPACE1：DB2 用来放置分组、排序、连接和重组中间结果的临时工作区域。
- USERSPACE1：默认情况下存放所有用户对象(表、索引)的地方。

在 Linux/UNIX 系统下，这些文件默认都放在实例用户的 home 目录下，在 Windows 系统中，这些文件默认都存放在 DB2 产品所在的安装盘的 DB2 目录下。对于简单的应用程序，这个默认配置应该可以满足需要。但是，我们可能希望改变数据库文件的位置，或者改变 DB2 管理这些对象的方式。接下来，我们将更详细地研究 CREATE DATABASE 命令。

对于从 DB2 V8 进行迁移的用户，有如下特殊的注意事项：在 DB2 V9 之前，CREATE DATABASE 命令会为上面列出的所有对象创建 SMS 表空间。在 DB2 V9 中，除了系统临时表空间外，所有表空间默认都将定义为自动存储(DMS)表空间。

2) CREATE DATABASE 命令

DB2 CREATE DATABASE 命令的完整语法很复杂，下面说明了 DBA 感兴趣的大多数选项。

```
>>-CREATE--+-DATABASE-+-database-name-+----------------------+-->
           '-DB-------'               '-| Database options |-'
```

数据库选项

```
CREATE DATABASE 选项:
 |--+-----------------------------+----------------------------->
   '-AUTOMATIC STORAGE--NO|YES--'
>--+-----------------------------------------+-------------->
   '-ON----+-path--+-+--+----------------------+-'
          '-drive-'    '-DBPATH ON--+-path--+-'
                                    '-drive-'
>--+----------------------+---------------------------------->
   '-ALIAS--database-alias-'
>--+-----------------------------------------+-------------->
   '-USING CODESET--codeset--TERRITORY--territory-'
```

```
    >--+--------------------------------------+-------------------------->
       |               .-SYSTEM---------. |
       '-COLLATE USING--+-COMPATIBILITY--+-'
                        +-IDENTITY-------+
                        +-IDENTITY_16BIT-+
     >--+-----------------------------------------+--------------------->
       '-CATALOG TABLESPACE--| tblspace-defn |-'
    >--+-------------------------------------+------------------------->
       '-USER TABLESPACE--| tblspace-defn |-'
    >--+------------------------------------------+-------------------->
       '-TEMPORARY TABLESPACE--| tblspace-defn |-'
```

表空间选项

```
    tblspace-defn:
     |--MANAGED BY----------------------------------------------------->
    >--+-SYSTEM
USING--(----'container-string'-+--)----------------------------------+-->
       +-DATABASE
USING--(----+-FILE---+--'container-string'--number-of-pages-+--)-+
       |                 '-DEVICE-'                                     |
       '-AUTOMATIC
STORAGE---------------------------------------------------------'
     >--+------------------------------+--------------------------------->
       '-EXTENTSIZE--number-of-pages-'
    >--+--------------------------------+------------------------------->
       '-PREFETCHSIZE--number-of-pages-'
    >--+----------------------+--+------------------------------------+->
       '-AUTORESIZE--+-NO--+-'  '-INITIALSIZE--integer--+-K|M|G-+-'
                     '-YES-'
    >--+-------------------------------------+------------------------->
       '-INCREASESIZE--integer--+-PERCENT-+-'
                                '-+-K|M|G-'
    >--+-------------------------------+------------------------------|
       '-MAXSIZE--+-NONE-------------+-'
                  '-integer--+-K|M|G-'
```

下面学习这些选项以及如何使用它们。

数据库位置

CREATE DATABASE 命令的参数之一是 ON path/drive 选项。这个选项告诉 DB2 希望在哪里创建数据库。如果没有指定路径，就会在数据库管理程序设置(DFTDBPATH 参数)

中指定的默认数据库路径上创建数据库。

```
test2:/home/db2inst4$db2 get dbm cfg | grep -i DFTDBPATH
 Default database path                       (DFTDBPATH) = /home/db2inst4
```

例如，以下 CREATE DATABASE 命令将数据库存放在 UNIX 操作系统的/db2/mydb 目录中：

```
CREATE DATABASE MYDB ON /db2/mydb
```

选择自动存储(默认设置)将允许 DBA 为数据库设置在创建所有表空间容器时可以使用的存储路径。DBA 不必显式地定义表空间的位置和大小，系统将自动地分配表空间。例如，下面的数据库创建语句将为数据库中的所有表空间设置自动存储：

```
CREATE DATABASE MYDB AUTOMATIC STORAGE YES ON
 /db2/mydbpath001,/db2/mydbpath002,/db2/mydbpath003
```

在 ON 选项后面，给出了 3 个文件目录(路径)。这 3 个路径是表空间容器的位置，数据库路径默认放在第一个路径下。当使用 AUTOMATIC STORAGE 定义表空间时，不需要提供其他参数：

```
CREATE TABLESPACE TEST MANAGED BY AUTOMATIC STORAGE;
```

在这个命令中，可以提供与表空间相关联的任何参数；虽然使用自动存储可以大大简化日常的表空间维护工作，但是与重要的大型生产表相关联的表空间可能需要 DBA 更多地干预。

在没有启用自动存储的数据库中创建表空间时，必须指定 MANAGED BY SYSTEM 或 MANAGED BY DATABASE 子句。使用这些子句会分别创建 SMS 表空间和 DMS 表空间。在这两种情况下，必须提供容器的显式列表。

如果数据库启用了自动存储，那么在定义表空间时还有另一个选择。可以指定 MANAGED BY AUTOMATIC STORAGE 子句，或者完全去掉 MANAGED BY 子句(这意味着自动存储)。在这种情况下，不提供容器定义，因为 DB2 会自动地分配容器。

代码页和整理次序

所有 DB2 字符数据类型(CHAR、VARCHAR、CLOB、DBCLOB)都有相关联的字符代码页。可以认为代码页是对照表，用来将字母数字数据转换为数据库中存储的二进制数据。一个 DB2 数据库只能使用一个代码页。代码页是在 CREATE DATABASE 命令中使用 CODESET 和 TERRITORY 选项设置的。代码页可以使用单一字节来表示字母数字字符(单一字节可以表示 256 个独特元素)，也可以使用多个字节。英语等语言包含的独特字符相当

少，因此单字节代码页(SBCS)对于存储数据足够了。东亚国家语言(中文、日文、韩文等)需要超过 256 个元素才能表示所有的独特字符，因此需要多字节代码页(通常是双字节代码页 DBCS)。

在默认情况下，数据库的整理次序根据 CREATE DATABASE 命令中使用的代码集进行定义。默认选项 COLLATE USING SYSTEM，会根据为数据库指定的 TERRITORY 对数据值进行比较。如果使用选项 COLLATE USING IDENTITY，那么以逐字节的方式使用二进制表示来比较所有值。

例如中文代码页为 1386，代码集为 GBK，TERRITORY 为 CN。创建数据库时要注意选择合适的代码页，这些参数在数据库创建好后都不能再进行修改，务必慎重选择。如果客户端访问数据库服务器时代码页不一样，将无法访问。

对于需要使用 XML 数据的应用程序，有如下特殊的注意事项。当前，DB2 只在定义为 Unicode 数据库才能同时存储 XML 文档和 SQL 数据的更多传统格式，比如整数、日期/时间、变长字符串等等。随后，你将在这个数据库中创建对象来管理 XML 和其他类型的数据。如果数据库在创建时没有启用 Unicode 支持，就不能在其中创建 XML 数据。

假如要创建同时支持 XML 和 SQL 的数据库，请执行如下命令：

```
create database xmldb using codeset UTF-8 territory us
```

一旦创建 Unicode 数据库，就不需要发出任何专门的命令或采取任何进一步措施来使 DB2 能够以自身分层的格式存储 XML 数据和关系数据。

表空间定义

3 个表空间(SYSCATSPACE、TEMPSPACE1、USERSPACE1)都是在默认目录中自动创建的(ON 关键字)，除非指定它们的位置。对于每个表空间，DBA 可以指定表空间应该使用的文件系统的特征。

3 个表空间使用以下语法进行定义：

```
>--+---------------------------------------+-------------------->
   '-CATALOG TABLESPACE--| tblspace-defn |-'
>--+-----------------------------------+---------------------->
   '-USER TABLESPACE--| tblspace-defn |-'
>--+-----------------------------------------+------------------>
   '-TEMPORARY TABLESPACE--| tblspace-defn |-'
```

如果省略任何关键字，DB2 将使用默认值来生成表空间。表空间定义采用这些选项，语法如下：

```
    |--MANAGED BY-------------------------------------------------->
                          |>--+-SYSTEM
USING--(----'container-string'-+--)---------------------------------+-->
       '-DATABASE
USING--(----+-FILE---+--'container-string'--number-of-pages-+--)-'
'-DEVICE-'
      >--+-----------------------------+------------------------------>
       '-EXTENTSIZE--number-of-pages-'
      >--+--------------------------------+--------------------------->
       '-PREFETCHSIZE--number-of-pages-'
```

注意，上面的语法不包括与自动存储数据库相关联的选项。

我们来详细看看这种语法。MANAGED BY 选项让 DB2 生成这些表空间并决定如何管理表空间。SMS 表空间使用 SYSTEM USING 关键字，如下所示：

```
SYSTEM USING ('container string')
```

对于 SMS 表空间，容器字符串(container string)标识一个或多个将属于这个表空间的容器，表空间数据将存储在这些容器中。每个容器字符串可以是绝对的或相对的目录名。如果目录名不是绝对的，就相对于数据库目录。如果目录的任何部分不存在，数据库管理程序就会创建这个目录。容器字符串的格式取决于操作系统。

使用 DATABASE USING 关键字定义 DMS 表空间：

```
DATABASE USING ( FILE/DEVICE 'container string' number of pages|K|M|G )
```

对于 DMS 表空间，容器字符串标识一个或多个将属于这个表空间的容器，表空间数据将存储在这些容器中。指定容器的类型(FILE 或 DEVICE)和大小(按照 PAGESIZE 大小的页面)。大小还可以指定为整数，后面跟着 K(表示千字节)、M(表示兆字节)或 G(表示千兆字节)。可以混合指定 FILE 和 DEVICE 容器。

对于 FILE 容器，容器字符串必须是绝对或相对的文件名。如果文件名不是绝对的，就相对于数据库目录。如果目录名的任何部分不存在，数据库管理程序就会创建这个目录。如果文件不存在，数据库管理程序就会创建这个文件并初始化为指定的大小。对于 DEVICE 容器，容器字符串必须是设备名而且这个设备必须已经存在；并且对于 DEVICE 容器，通常需要使用操作系统 root 权限创建逻辑卷并且赋予 DB2 实例使用的权限，一般通过 UNIX/Linux 的 chown 命令来实现这一点。

重要提示：所有容器必须在所有数据库中唯一；一个容器只能属于一个表空间。

```
EXTENTSIZE  number of pages
```

EXTENTSIZE 指定数据库可以写到容器中的 PAGESIZE 页面数量，达到这个数量之后将跳到下一个容器。对于含有多个容器的表空间，DB2 以循环方式使用这些容器，以 EXTENTSIZE 为切换容器的单位，即先写满第一个容器的 EXTENT，再写第二个容器的一个 EXTENT，依次执行。EXTENTSIZE 的值还可以指定为整数，后面跟着 K、M 或 G，EXTENTSIZE 必须是 PAGESIZE 的整数倍。EXTENTSIZE 的大小是在表空间级定义的。一旦为表空间指定扩展数据块大小，就不能改变了。数据库配置参数 DFT_EXTENT_SZ 指定数据库中所有表空间的默认扩展数据块大小。这个值的范围是 2 到 256 个页面；因此，绝对大小是从 8KB 到 1024KB(对于 4KB 页面)，或者从 16 KB 到 2048KB(对于 8KB 页面)。可以在 CREATE TABLESPACE 语句中使用 EXTENTSIZE 参数覆盖这个数字。

```
PREFETCHSIZE  number of pages
```

PREFETCHSIZE 指定在执行数据预获取时将从表空间中读取的 PAGESIZE 页面数量。连续的预读取是指数据库管理程序能够提前预测查询，在实际引用页面之前读取这些页面。这样查询就不需要等待底层操作系统执行 I/O 操作。这种异步的检索可以显著减少执行时间。可以通过修改 CREATE TABLESPACE 语句中的 PREFETCHSIZE 参数来控制执行预获取的大小。在默认情况下，这个值设置为 DFT_PREFETCH_SZ 数据库配置参数，这个值代表在 DB2 触发预读取请求时每次读取多少个页面。通过将这个值设置为扩展数据块大小的倍数，可以并行地读取多个扩展数据块。当表空间的容器在不同的硬盘上时，这个功能甚至效率更高。预读取大小还可以指定为整数，后面跟着 K、M 或 G。

关于 PREFETCHSIZE 的设置，我们在后面介绍表空间性能时还会详细讲解。

CREATE DATABASE 命令示例

下面是 CREATE DATABASE 命令示例，这里使用了前面讨论的许多选项：

```
CREATE DATABASE MYDB
DFT EXTENT SZ 4
CATALOG TABLESPACE MANAGED BY DATABASE USING
  (FILE '/data1/CATALOG.DAT' 20000, FILE '/data2/CATALOG.DAT' 20000)
    EXTENTSIZE 8
   PREFETCHSIZE 16
TEMPORARY TABLESPACE MANAGED BY SYSTEM USING
  ('/data1/TEMPTS1','/data2/TEMPTS2')
USER TABLESPACE MANAGED BY DATABASE USING
   (FILE '/data1/USERTS.DAT' 1200)
   EXTENTSIZE 24
   PREFETCHSIZE 48
```

我们来详细地看看每一行：

- CREATE DATABASE：这条语句定义要创建的数据库的名称。
- DFT_EXTENT_SZ 4：这个参数告诉 DB2 默认的扩展数据块大小是 4 个页面，除非在创建表空间时显式地声明，否则默认使用这个值。
- CATALOG TABLEPSACE MANAGED BY DATABASE USING：DB2 编目空间将由数据库管理。
- FILE '/data…'：数据库编目表空间的位置将跨两个文件，每个文件有 20000 个页面的空间。
- EXTENTSIZE 8：EXTENTSIZE 是 8 个页面。这个设置会覆盖 DFT_EXTENT_SZ。
- PREFETCHSIZE 16：在查询处理期间，同时预读取 16 个页面。
- TEMPORARY TABLESPACE MANAGED BY SYSTEM USING：DB2 使用的临时空间将由操作系统处理。
- 'TEMPTS..' ...：临时空间将跨两个文件，文件的大小在 DB2 执行期间自动地调整。
- USER TABLESPACE MANAGED BY DATABASE USING：用户表空间(放置真正的表的地方)将由 DB2 直接管理。
- EXTENTSIZE 24：USER 表空间的 EXTENTSIZE 是 24 个页面。
- PREFETCHSIZE 48：查询处理期间，同时预读取 48 个页面。

上面介绍了关于如何创建 DB2 数据库的背景知识。在大多数情况下，CREATE DATABASE 命令的默认值提供了可以满足开发和测试需要的数据库。一旦决定将数据库转入生产环境，就需要对 DB2 使用的数据布局和表空间定义付出更大的努力。尽管这需要做更多的规划工作，但是产生的数据库更容易管理，性能也可能更好。关于这部分内容我们会在 3.2 节中详细讲解。

2. 使用创建数据库向导创建数据库

如果觉得通过上面命令创建数据库比较麻烦，在 Windows 平台上，可以通过创建数据库向导来创建数据库，创建数据库向导将带领我们执行许多步骤来生成数据库。向导首先询问数据库的名称、创建数据库时的默认驱动器(如果没有指定其他驱动器，就会使用这个驱动器)和别名，如图 3-7 所示。另外，可以添加关于数据库内容的注释。

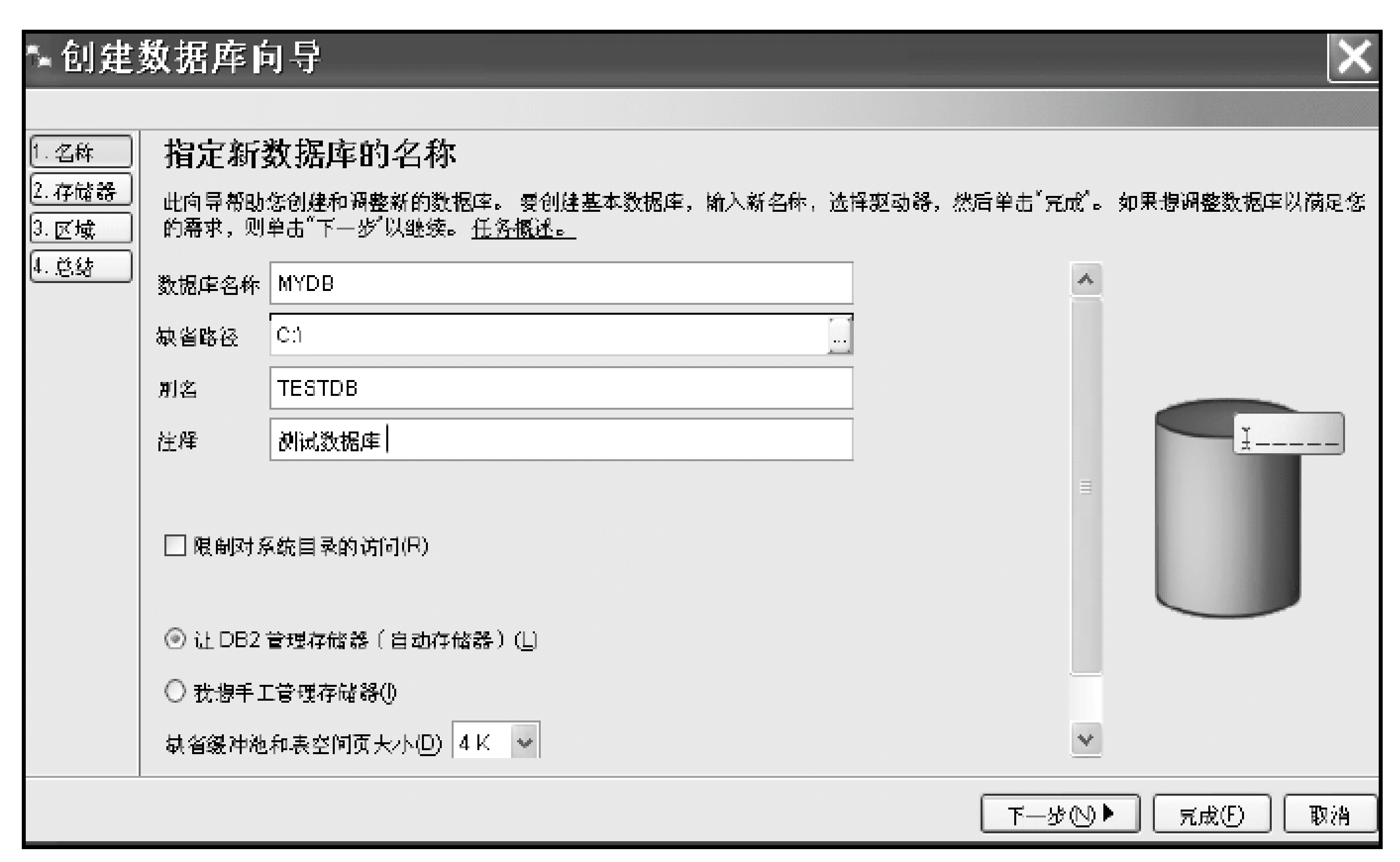


图 3-7　指定数据库的名称、默认路径、别名等

关于图 3-7 有几点需要特别注意。如果希望在数据库中使用 XML 列，就必须定义为 UTF-8(*Enable database for XML*)。另外在 DB2 V9 中，自动存储是数据库的默认设置。如果希望覆盖这个默认设置，就必须选择“我想手工管理存储器”选项。

创建数据库向导：用户/编目/临时表

选择“我想手工管理存储器”选项后，向导的后 3 个对话框要求填写关于如何创建用户、编目和临时表空间的信息。如果选择“低维护”选项，向导就会创建 SMS 表空间。如果选择“高性能”，就需要指定用于这个表空间的设备和文件系统，如图 3-8 所示。

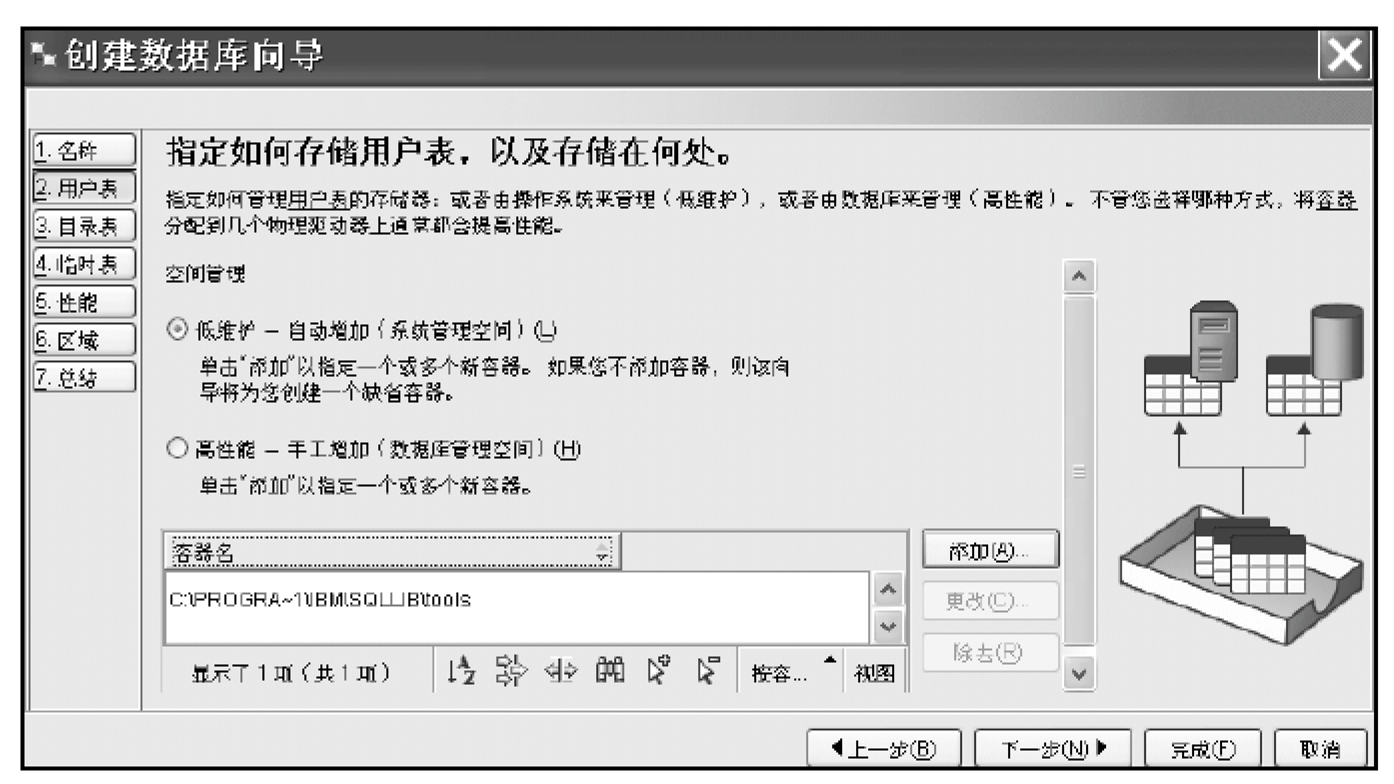


图 3-8　指定如何创建用户、编目和临时表空间

无论选择哪个选项，都可以指定希望分配给这个表空间的容器(文件或设备)。如果单

击“添加”按钮，将会显示另一个对话框，如图 3-9 所示，可以在这里定义要使用的容器。



图 3-9　定义容器

如果没有为表空间指定容器或文件，DB2 将在前面指定的默认驱动器上自动地生成一个。

创建数据库向导：性能选项

可以设置两个性能参数——扩展数据块大小(EXTENTSIZE)和预取大小(PREFETCHSIZE)，如图 3-10 所示。



图 3-10　调整数据库的性能

我们来看看这两个参数的作用：

- EXTENTSIZE：扩展数据块(extent)是表空间容器中的空间单元。数据库对象(除了 LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB 或 DCLOB)都存储在

DB2 的页面中。这些页面组合成扩展数据块。扩展数据块的大小是在表空间级定义的。一旦为表空间指定扩展数据块的大小，就不能改变了。数据库配置参数 DFT_EXTENT_SZ 指定数据库中所有表空间的默认扩展数据块大小。这个值的范围是 2 到 256 个页面；因此，绝对大小是从 8KB 到 1024KB(对于 4KB 页面)，或者从 16KB 到 2048KB(对于 8KB 页面)。可以在 CREATE TABLESPACE 语句中使用 EXTENTSIZE 参数覆盖这个数字。

如果打算在表的设计中使用多维聚簇(MDC)，扩展数据块就是重要的设计决定之一。MDC 表将为创建的每个新的维集分配扩展数据块。如果扩展数据块太大，那么扩展数据块的很大一部分有可能是空的(对于包含很少记录的维集)。关于 MDC 及其对 EXTENTSIZE 影响的更多信息，请参考《高级进阶 DB2(第 2 版)》一书。

- PREFETCHSIZE：连续的预读取是指数据库管理程序能够提前预测查询，在实际引用页面之前读取这些页面。这种异步的检索可以显著减少执行时间。可以通过修改 CREATE TABLESPACE 语句中的 PREFETCHSIZE 参数来控制执行预读取的积极程度。在默认情况下，这个值设置为 DFT_PREFETCH_SZ 数据库配置参数。这个值代表在 DB2 触发预获取请求时每次读取多少个页面。通过将这个值设置为扩展数据块大小的倍数，可以并行地读取多个扩展数据块。当表空间容器在不同的硬盘上时，这个功能甚至效率更高。

这些参数的默认值对于许多应用程序是合适的，但是对于执行许多查询或分析大量数据的应用程序，可以考虑设置更高的 PREFETCHSIZE。

创建数据库向导：代码页和整理次序

在数据库创建过程中，遇到的下一个选项涉及代码页和整理次序，如图 3-11 所示。

当将 DB2 应用程序绑定到 DB2 数据库时，会对应用程序和数据库的代码页进行比较。如果它们的代码页不相同，就会尝试对每条 SQL 语句执行代码页转换。如果使用与访问的数据库不同的代码页，那么一定要确保代码页是兼容的并且可以执行转换。

在默认情况下，数据库的整理次序根据 CREATE DATABASE 命令中使用的编码集进行定义。如果指定选项 COLLATE USING SYSTEM，就根据为数据库指定的 TERRITORY 对数据值进行比较。如果使用选项 COLLATE USING IDENTITY，那么以逐字节的方式使用二进制表示来比较所有值。在需要以本机(二进制)格式存储数据时，要避免使用有代码页的数据类型。一般情况下，使用相同的应用程序代码页和数据库代码页是有好处的，可以避免进行代码页转换。

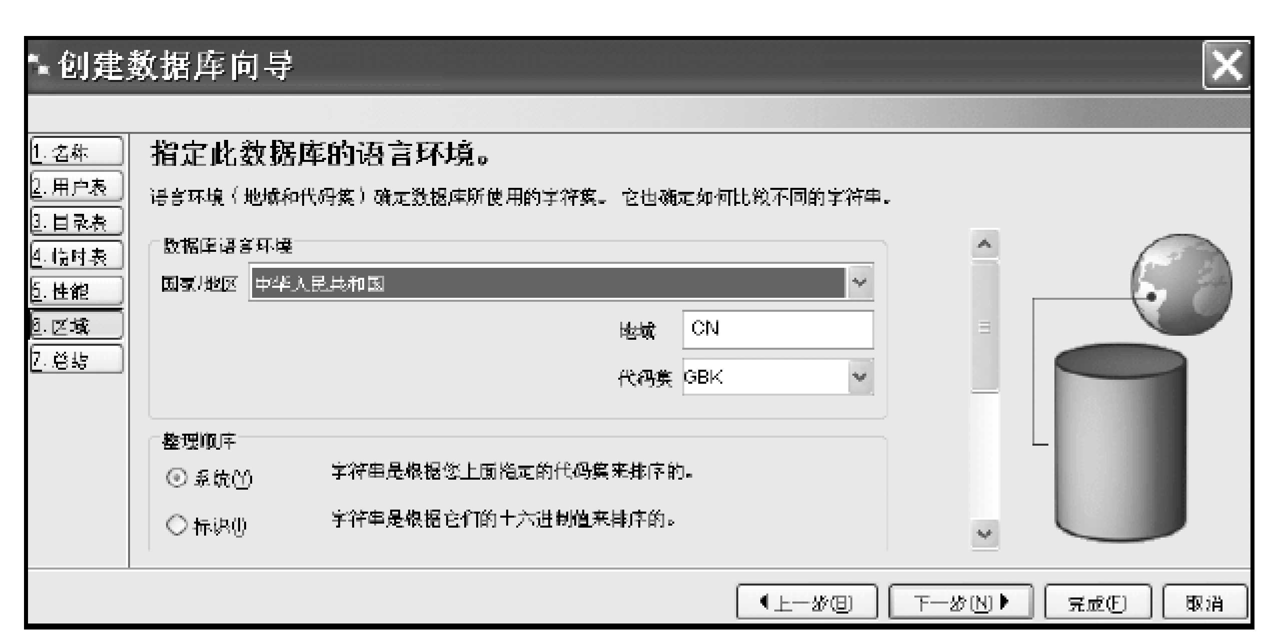


图 3-11　指定数据库的语言环境

创建数据库向导：创建总结

在设置好所有参数之后，创建数据库向导会显示总结页面，其中总结了你做出的所有选择，如图 3-12 所示。



图 3-12　查看数据库创建总结

总结页面上一个极其有用的特性是“显示命令”按钮。如果单击，就会看到用来创建数据库的 DB2 命令，如图 3-13 所示。

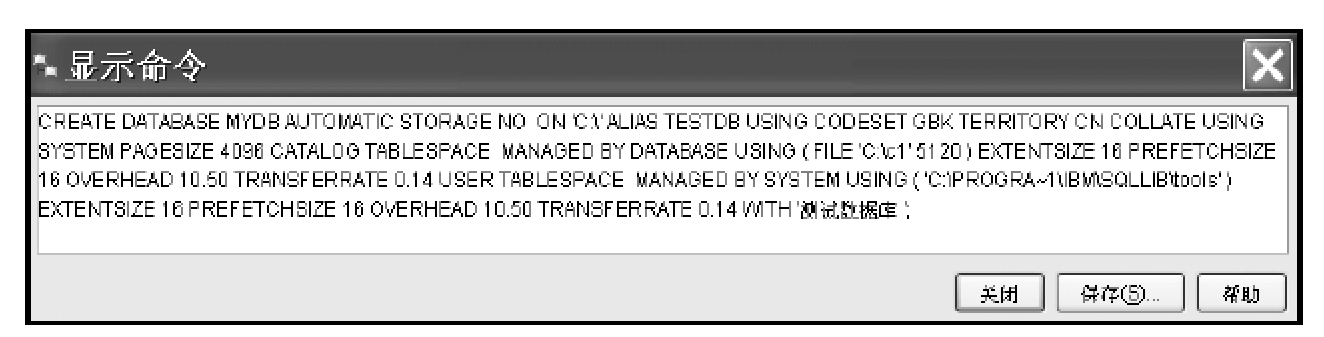


图 3-13　显示数据库的创建命令

可以保存这个命令以便在以后执行，或者将它复制并粘贴到正在开发的脚本中。如果对输入到系统中的参数感到满意，就单击“完成”按钮来创建数据库。

3.1.4 数据库目录

我们在第 2 章创建实例时讲过，当创建实例时，就会生成实例目录。同样，当创建数据库时，关于数据库的信息(包括默认信息)会存储在目录层次结构中，这就是数据库目录。此分层目录结构的创建位置取决于你在 CREATE DATABASE 命令中提供的路径信息。如果在创建数据库时未指定目录路径或驱动器的位置，那么将使用默认位置。建议创建数据库时明确数据库路径。数据库目录中存放的是数据库表空间、表、索引、容器等信息，这个目录至关重要，一定要注意它的安全性。

在 CREATE DATABASE 命令中，在指定为数据库路径的目录中，将创建使用实例名的子目录。这个子目录确保在同一目录下的不同实例中创建的数据库不会使用相同的路径。在实例名子目录下面，将创建名为 NODE0000 的子目录。这个子目录可以区分逻辑分区数据库环境中的数据库分区。在节点名目录下面，将创建名为 SQL00001 的子目录。此子目录的名称使用了数据库标记并表示正在创建的数据库。SQL00001 包含与第一个创建的数据库以及随后创建的具有更高编号(SQL00002 等)的数据库相关联的对象。这些子目录可以区分在 CREATE DATABASE 命令中，在指定目录的实例中创建的数据库。

目录结构如下所示：

```
<your_database_path>/<your_instance>/NODE0000/SQL00001/
```

详细的信息如图 3-14 所示。

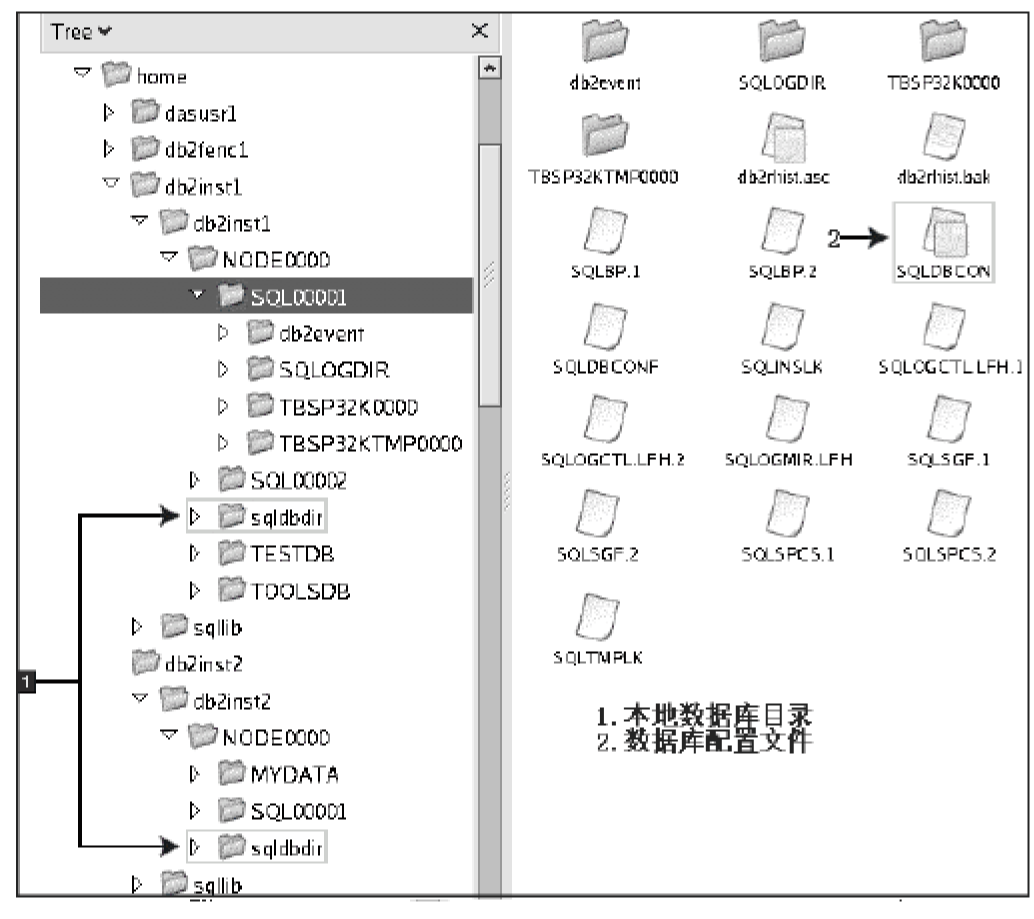


图 3-14 数据库目录

数据库目录中包含下列作为 CREATE DATABASE 命令一部分进行创建的文件：

- 文件 SQLBP.1 和 SQLBP.2 中包含缓冲池信息。这两个文件互为副本以实现备份。

- SQLSPCS.1 和 SQLSPCS.2 文件中包含表空间信息。这两个文件互为副本以实现备份。
- 文件 SQLSGF.1 和 SQLSGF.2 包含与数据库自动存储有关的存储路径信息。这两个文件互为副本以实现备份。
- SQLDBCONF 文件中包含数据库配置信息。切勿编辑此文件。

注意：

SQLDBCON 文件在先前发行版中使用，并且包含在 SQLDBCONF 损坏时可以使用的类似信息。

要更改配置参数，请使用 UPDATE DATABASE CONFIGURATION 和 RESET DATABASE CONFIGURATION 语句。

- DB2RHIST.ASC 历史记录文件及其备份 DB2RHIST.BAK 中包含关于备份、复原、表装入、表重组、表空间改变和其他数据库更改的历史记录信息。
  DB2TSCHG.HIS 文件中包含日志文件级别的表空间更改的历史记录。对于每个日志文件，DB2TSCHG.HIS 中包含有助于标识日志文件影响哪些表空间的信息。表空间恢复使用此文件中的信息来确定在进行表空间恢复期间要处理哪些日志文件。可以在文本编辑器中检查这两个历史记录文件中的内容。
- 日志控制文件 SQLOGCTL.LFH.1 及其镜像副本 SQLOGCTL.LFH.2 和 SQLOGMIR.LFH 中包含有关活动日志的信息。崩溃恢复处理过程使用这些文件中的信息来确定要在日志中后退多远来开始崩溃恢复。SQLOGDIR 子目录包含实际的日志文件。

注意：

应确保不要将日志子目录映射到用于存储数据的磁盘。这样一来，在磁盘发生问题时，只会影响到数据或日志，而不会同时影响这两者。由于日志文件与数据库容器不会争用同一磁盘磁头的移动，因此这可提供很多性能方面的好处。要更改日志子目录的位置，请更改 *newlogpath* 数据库配置参数。这部分内容会在《DB2 数据库性能调整与优化(第 2 版)》的第 6 章中讲解。

- SQLINSLK 文件用于确保数据库只能由数据库管理器的单个实例使用。
- 在创建数据库的同时，还创建了详细的死锁事件监视器。详细的死锁事件监视器文件存储在目录节点的数据库目录中，名为 db2detaildeadlock。

在非自动存储数据库中，SMS 数据库目录的其他信息

在非自动存储数据库中，SQLT*子目录包含运作数据库所需的默认“系统管理的空间”(SMS)表空间。创建数据库时会生成 3 个默认表空间：

- SQLT0000.0 子目录中包含带有系统目录表的目录表空间
- SQLT0001.0 子目录中包含默认临时表空间
- SQLT0002.0 子目录中包含默认用户数据表空间

每个子目录或容器中都会创建名为 SQLTAG.NAM 的文件。这个文件可以标记正在使用中的子目录，因此在以后创建其他表空间时，不会尝试使用这些子目录。

此外，名为 SQL*.DAT 的文件中还存储有关于目录或容器包含的每个表的信息。星号(*)将被唯一的一组数字取代，用来识别每个表。对于每个 SQL*.DAT 文件，可能有一个或多个下列文件，这取决于表类型、表的重组状态或者表是否存在索引、LOB 或 LONG 字段：

- SQL*.BKM(如果是 MDC 表，那么包含块分配信息)
- SQL*.LF(包含 LONG VARCHAR 或 LONG VARGRAPHIC 数据)
- SQL*.LB(包含 BLOB、CLOB 或 DBCLOB 数据)
- SQL*.XDA(包含 XML 数据)
- SQL*.LBA(包含关于 SQL*.LB 文件的分配和可用空间信息)
- SQL*.INX(包含索引表数据)
- SQL*.IN1(包含索引表数据)
- SQL*.DTR(包含用于重组 SQL*.DAT 文件的临时数据)
- SQL*.LFR(包含用于重组 SQL*.LF 文件的临时数据)
- SQL*.RLB(包含用于重组 SQL*.LB 文件的临时数据)
- SQL*.RBA(包含用于重组 SQL*.LBA 文件的临时数据)

如果创建了多个数据库，那么可以通过 db2 list db directory on dbpath 查看每个数据库的目录。

数据库目录对于应用和数据库用户来说是透明的，他们看到的是数据库逻辑层面的表、索引等对象。而数据库目录是面向 DBA 的，所以 DBA 了解数据库的物理存储模型和逻辑存储模型。逻辑模型和物理模型是用系统编目表来统一的。

3.2　设计表空间

3.2.1　创建表空间

表空间建立数据库系统使用的物理存储设备与用来存储数据的逻辑对象或表之间的关系。我们在前面创建数据库部分讲解了表空间的类型，对于非自动存储表空间，在创建表空间时，必须知道将引用的容器的设备名或文件名。另外，必须知道要分配给表空间的每个设备名或文件名及分配空间大小。对于自动存储表空间，数据库管理器将根据与数据库关联的存储路径将容器指定给表空间。

在数据库内创建表空间，会将容器分配到表空间，并在数据库系统目录表中记录定义和属性，然后就可以在此表空间内创建表。当创建数据库时，会创建 3 个初始表空间。这 3 个初始表空间的页大小基于使用 CREATE DATABASE 命令时建立或接受的默认值。此默认值还表示所有将来 CREATE BUFFERPOOL 和 CREATE TABLESPACE 语句的默认页大小。如果在创建数据库时不指定页大小，那么默认页大小是 4KB。如果在创建表空间时不指定页大小，那么默认页大小是创建数据库时设置的页大小。

为了创建表空间，可以通过控制中心或命令行创建。使用控制中心创建表空间的界面如图 3-15 所示。

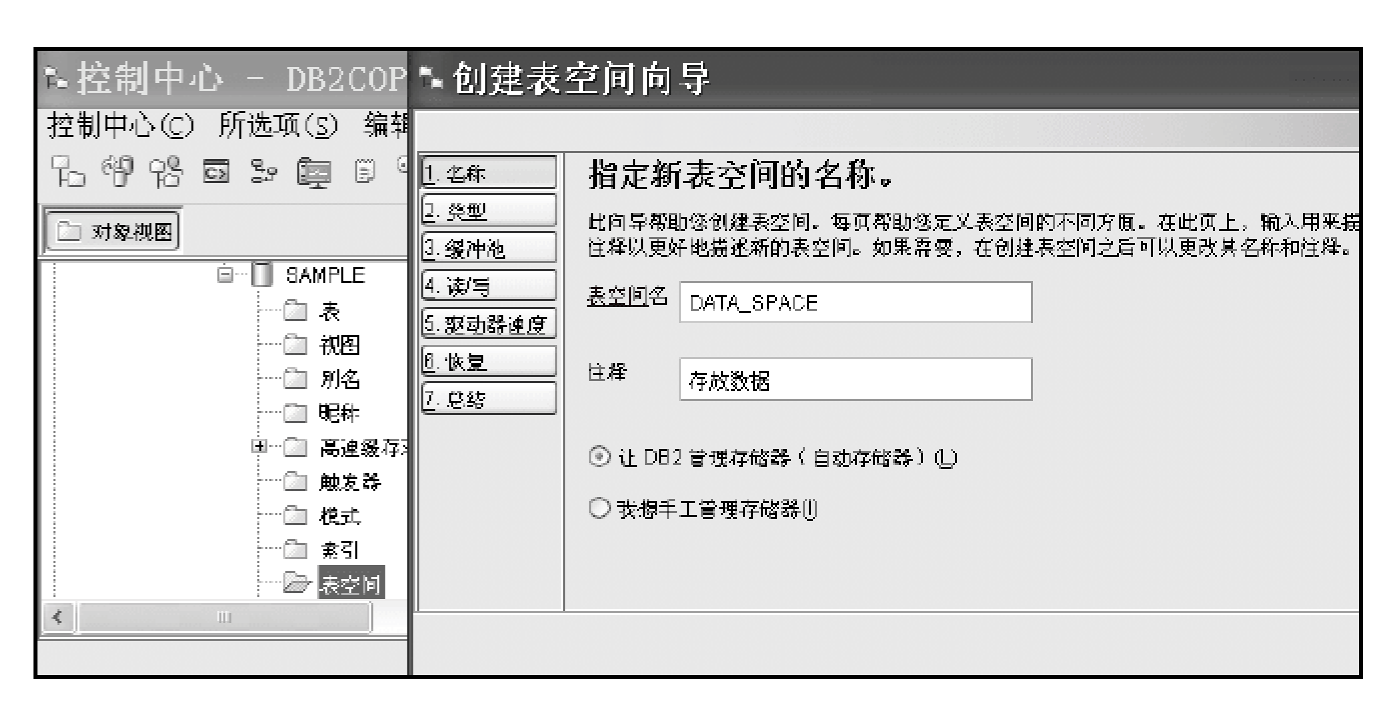


图 3-15　使用控制中心创建表空间

用图形化界面创建表空间比较简单，下面重点讲解如何使用命令行创建表空间。

1. 创建用户表空间

要使用命令行创建 SMS 表空间，请输入：

```
CREATE TABLESPACE <NAME> MANAGED BY SYSTEM  USING ('<path>')
```

要使用命令行创建 DMS 表空间，请输入：

```
    CREATE TABLESPACE <NAME>   MANAGED BY DATABASE   USING (DEVICE |FILE'<path>'
<size>)
```

要使用命令行创建自动存储表空间，请输入下列任一语句：

```
    CREATE TABLESPACE <NAME>
    CREATE TABLESPACE <NAME> MANAGED BY AUTOMATIC STORAGE
```

通过使用 3 个不同的驱动器上的 3 个目录，下列 SQL 语句在 UNIX 中创建了一个 SMS 表空间：

```
    CREATE TABLESPACE TS1 MANAGED BY SYSTEM USING ('/data1/nxz tbsp',
'/data2/nxz_tbsp', '/data3/nxz_tbsp')
```

以下 SQL 语句使用各自有 5000 页的两个文件容器创建了一个 DMS 表空间：

```
    CREATE TABLESPACE TS2 MANAGED BY DATABASE
    USING (FILE '/data1/acc_tbsp' 5000, FILE '/data2/acc_tbsp' 5000)
```

注意在创建 DMS 表空间时，表空间文件容器不需要创建，DB2 会自动创建(裸设备容器无法自动创建，需要 root 用户参与)。

在前面两个示例中，为表空间容器提供了显式的名称。但是，如果指定相对容器名，那么将在为数据库创建的子目录中创建容器。

在创建表空间容器时，数据库管理器会创建任何不存在的目录和文件。例如，如果将容器指定为/prod/user_data/container1，而目录/prod 不存在，那么数据库管理器会创建目录/prod 和/prod/user_data。

在 Linux/UNIX 中，数据库管理器创建的任何目录都是使用权限位 711 创建的，这意味着只有实例所有者才拥有读写访问权和执行访问权。因为只有实例所有者具有这种访问权，所以当正在创建多个实例时，可能会出现下列情况：

- 使用与上面描述的相同的目录结构，假定目录级别/prod/user_data 不存在。
- user1 创建实例(默认情况下命名为 user1)，接着创建数据库，然后创建表空间，并且/prod/user_data/container1 将作为表空间的容器。
- user2 创建实例(默认情况下命名为 user2)，接着创建数据库，然后尝试创建表空间，并且/prod/user_data/container2 将作为表空间的容器。

因为数据库管理器根据第一个请求使用权限位 700 创建了目录级别/prod/user_data，所以 user2 没有对这些目录级别的访问权，因此不能在这些目录中创建 container2。在这种情况下，CREATE TABLESPACE 操作将失败。

解决此冲突有两种方法：

- 在创建表空间之前创建目录/prod/user_data，并将许可权设置为 user1 和 user2 创建表空间所需的任何访问权。如果所有级别的表空间目录都存在，那么数据库管理器不会修改访问权。
- 在 user1 创建/prod/user_data/container1 之后，将/prod/user_data 的许可权设置为 user2 创建表空间所需的任何访问权。

如果数据库管理器创建了子目录，那么在删除表空间时，数据库管理器也可能将子目录删除。

下列 SQL 语句在 AIX 系统中创建了使用具有 10 000 页的 3 个裸设备作为表空间容器的 DMS 表空间，并指定它们的 I/O 特征：

```
    CREATE TABLESPACE TS1  MANAGED BY DATABASE
     USING (DEVICE '/dev/rdblv6' 10000, DEVICE '/dev/rdblv7' 10000, DEVICE
'/dev/rdblv8' 10000) OVERHEAD 7.5  TRANSFERRATE 0.06
```

在此 SQL 语句中提到的裸设备必须已经存在，并且实例所有者和 SYSADM 组必须能够写入这些设备。

还可以创建表空间，使用的页大小比默认的 4KB 更大。下列 SQL 语句在 Linux 和 UNIX 系统中创建具有 8KB 页大小的 SMS 表空间：

```
    CREATE TABLESPACE SMS8K  PAGESIZE 8192  MANAGED BY SYSTEM
              USING ('FSMS_8K_1')   BUFFERPOOL BUFFPOOL8K
```

注意，相关联的缓冲池也必须具有相同的 8KB 页大小，而且只有在激活了由创建的表空间引用的缓冲池之后才能使用该表空间。

2. 创建系统临时表空间

系统临时表空间用来存储分组、排序、连接、重组、创建索引操作等中间结果。数据库必须始终至少有一个这样的表空间。创建数据库时，定义的 3 个默认表空间之一便是名为“TEMPSPACE1”的系统临时表空间。

要创建另一个系统临时表空间，可使用 CREATE TABLESPACE 语句。例如：

```
    CREATE SYSTEM TEMPORARY TABLESPACE tmp tbsp
    MANAGED BY SYSTEM USING ('/data1/tmp_tbsp','/data2/tmp_tbsp')
```

对于每个页大小，至少应具有一个和该页大小匹配的系统临时表空间。

3. 创建用户临时表空间

用户临时表空间不是在创建数据库时默认创建的。如果应用程序需要使用临时表，那么需要创建将驻留临时表的用户临时表空间。用户临时表空间通常用来批量插入、批量更新和批量删除以加快速度。

使用 DECLARE GLOBAL TEMPORARY TABLE 语句声明临时表时，必须要求用户临时表空间存在。

要创建用户临时表空间，可使用 CREATE TABLESPACE 语句：

```
CREATE USER TEMPORARY TABLESPACE usr tbsp  MANAGED BY DATABASE
USING (FILE '/data1/user_tbsp' 5000,  FILE '/data2/user_tbsp' 5000)
```

3.2.2 维护表空间

1. 查看表空间

可以使用 DB2 LIST TABLESPACES[SHOW DETAIL]来查看表空间的详细信息。

LAST TABLESPACES 命令的输出信息如下：

```
                        Tablespaces for Current Database
Tablespace ID                        = 0
Name                                 = SYSCATSPACE
Type                                 = System managed space
Contents                             = Any data
State                                = 0x0000
  Detailed explanation:    Normal
Tablespace ID                        = 1
Name                                 = TEMPSPACE1
Type                                 = System managed space
Contents                             = System Temporary data
State                                = 0x0000
  Detailed explanation:    Normal
Tablespace ID                        = 2
Name                                 = USERSPACE1
Type                                 = System managed space
Contents                             = Any data
State                                = 0x0000
  Detailed explanation:    Normal
```

上面所示的这 3 个表空间是通过 CREATE DATABASE 命令自动创建的。用户可以通过在该命令中定制表空间选项来覆盖默认的表空间创建选项。但是在创建数据库时必须创建系统编目表空间和至少一个常规表空间，以及至少一个系统临时表空间。通过使用

CREATE DATABASE 命令或以后使用 CREATE TABLESPACE 命令，可以创建更多的所有类型的表空间(系统表空间除外)。在上述 3 个表空间中，系统编目表空间和系统临时表空间都是只读的，用户不可以在上面创建用户表，如下所示。

```
    test2:/home/db2inst4$db2 "create table t(i int) in  SYSCATSPACE"
    DB21034E  The command was processed as an SQL statement because it was not
a valid Command Line Processor command.  During SQL processing it returned:
SQL0287N  SYSCATSPACE cannot be used for user objects.  SQLSTATE=42838
```

查看表空间及容器的属性

指定 LIST TABLESPACES 命令的 SHOW DETAIL 选项将显示其他信息：

```
    LIST TABLESPACES SHOW DETAIL
```

默认情况下，将列出创建数据库时创建的那 3 个表空间。LIST TABLESPACES SHOW DETAIL 命令的输出信息如下：

```
    Tablespaces for Current Database
    Tablespace ID                      = 2
    Name                               = USERSPACE1
    Type                               = Database managed space
    Contents                           = Any data
    State                              = 0x0000
      Detailed explanation:    Normal
    Total pages                        = 25000---------总页数
    Useable pages                      = 24904--------可用页数
    Used pages                         = 336--------使用页数
    Free pages                         = 24568----空闲页数
    High water mark (pages)            = 336
    Page size (bytes)                  = 4096
    Extent size (pages)                = 32
    Prefetch size (pages)              = 16
    Number of containers               = 1
```

要列出容器，需要使用以上输出中的 Tablespace ID：

```
    LIST TABLESPACE CONTAINERS FOR 2
```

为了查看表空间容器的情况，可以使用 LIST TABLESPACE CONTAINERS 命令：

```
              Tablespace Containers for Tablespace 2

     Container ID                        = 0
     Name                               =
```

```
/db2/mydb/db2inst4/NODE0000/MYDB/T0000002/C0000000.LRG
     Type                                = File
```

该命令将列出指定表空间中的所有容器。如上所示的路径指向容器物理上所在的位置。

表空间状态

为了查看数据库中表空间的状态，可以使用命令：

```
    list tablespaces show detail
```

表空间可以有多种不同的状态，如下所示：

0x0	Normal
0x1	Quiesced：SHARE
0x2	Quiesced：UPDATE
0x4	Quiesced：EXCLUSIVE
0x8	Load pending
0x10	Delete pending
0x20	Backup pending
0x40	Rollforward in progress
0x80	Rollforward pending
0x100	Restore pending
0x100	Recovery pending(not used)
0x200	Disable pending
0x400	Reorg in progress
0x800	Backup in progress
0x1000	Storage must be defined
0x2000	Restore in progress
0x4000	Offline and not accessible
0x8000	Drop pending
0x2000000	Storage may be defined
0x4000000	StorDef is in 'final' state
0x8000000	StorDef was changed prior to rollforward
0x10000000	DMS rebalancer is active
0x20000000	TBS deletion in progress
0x40000000	TBS creation in progress
0x8	For service use only

关于表空间状态的详细信息，请读者参考“第 15 章：DB2 常见问题总结”。

2. 修改表空间

可使用控制中心或命令行来改变表空间。要使用命令行改变表空间，可使用 ALTER TABLESPACE 语句。可以改变 SMS、DMS 和自动存储容器，还可以重命名表空间，并将

表空间从脱机方式切换至联机方式。

对于 SMS 表空间，我们只能增加容器；对于 DMS 表空间，可以添加、扩展、重新平衡、删除或减少容器，或者调整容器大小。我们重点讲解 DMS 表空间的修改。使用控制中心修改表空间的界面如图 3-16 所示。

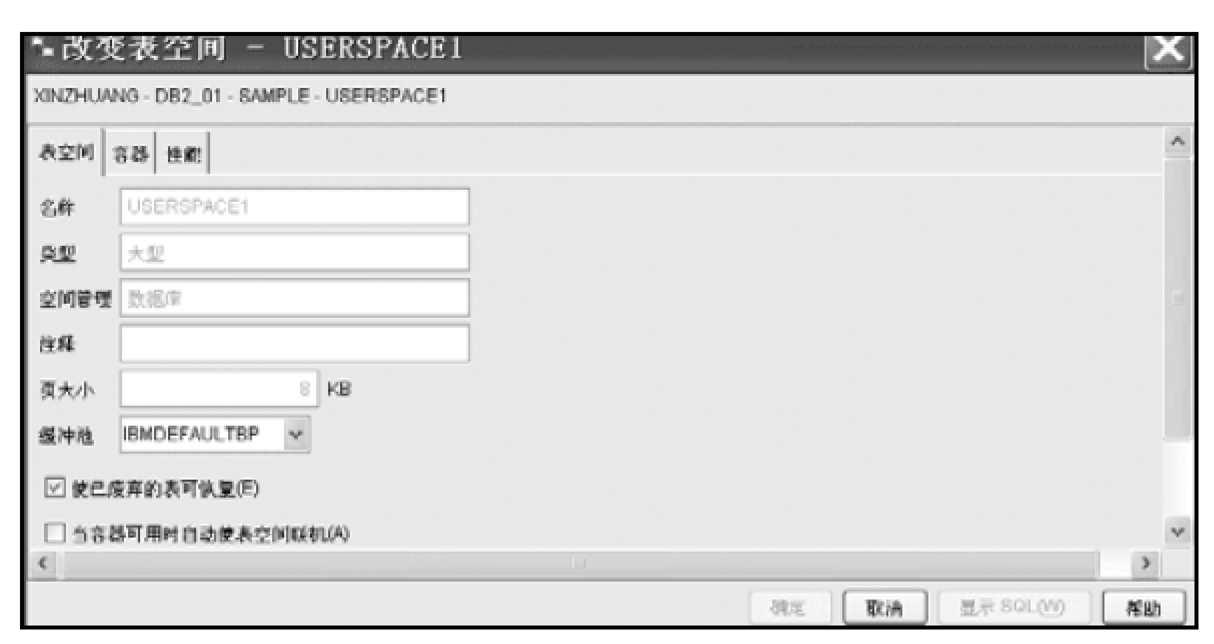


图 3-16　使用控制中心修改表空间

下面重点讲解如何使用命令行修改表空间。

添加或扩展 DMS 容器

通过将一个或多个容器添加至 DMS 表空间(即使用 MANAGED BY DATABASE 子句创建的表空间)，可以增大 DMS 表空间的大小。

当将新容器添加到表空间或扩展现有容器时，可能会发生表空间重新平衡(rebanlance)。重新平衡过程涉及将表空间扩展数据块从一个位置移至另一位置。在此过程中，将尝试在表空间内分割数据。重新平衡不必在所有容器上进行，但这取决于许多因素，例如现有容器的配置、新容器的大小和表空间满的程度。

在重新平衡期间，不限制对表空间的访问。如果需要添加多个容器，那么应该同时添加这些容器以减少重新平衡的次数。虽然重新平衡期间表空间仍然可以访问，但我们还是尽量避免在业务高峰期间增加容器，因为数据重新平衡期间系统中有很多的 I/O 活动。关于表空间重新平衡，其实 DB2 还有一些高级选项，但这部分内容超出了本书讲解范围。如果读者感兴趣，可以参考《高级进阶 DB2(第 2 版)》一书。

要使用命令行将容器添加到 DMS 表空间，请输入以下内容：

```
ALTER TABLESPACE <name>  ADD (DEVICE '<path>' <size>, FILE '<filename>' <size>)
```

以下示例说明如何将两个新设备容器(各含 10 000 页)添加到 Linux 和 UNIX 系统的表空间中：

```
ALTER TABLESPACE TS1  ADD (DEVICE '/dev/rhd9'  10000,  DEVICE  '/dev/rhd10' 10000)
```

添加容器会涉及表空间容器的重新平衡，如果不想这样，可以使用表空间扩展来修改容器大小，因为 extend 不会重新平衡表空间数据。

以下示例说明如何将所有容器扩展 10 000 页(各含 10 000 页)后添加到 Linux 和 UNIX 系统的表空间中：

```
ALTER TABLESPACE TS1 EXTEND (ALL 10000)
```

调整 DMS 容器的大小

不能手动调用自动存储表空间中容器的大小，否则将报错，如下所示：

```
test2:/$db2 "alter tablespace userspace1 extend (all 20)"

DB21034E  The command was processed as an SQL statement because it was not
a valid Command Line Processor command.  During SQL processing it returned:

SQL20318N  Table space "USERSPACE1" of type "AUTOMATIC STORAGE" cannot be
altered using the "EXTEND" operation.  SQLSTATE=42858
```

只能将每个操作系统裸设备用作容器。创建裸设备之后，其大小是固定的。当考虑使用调整大小或扩展选项来增大裸设备容器时，应先用操作系统命令检查裸设备大小以确保使用 ALTER TABLESPACE 命令并未将裸设备容器大小增大到大于裸设备大小。

要缩小现有容器的大小，可使用 RESIZE 或 REDUCE 选项。使用 RESIZE 选项时，作为语句一部分列示的所有容器都必须增大或减小大小。不能在同一条语句中增大某些容器而缩小其他容器。如果知道容器大小的新下限，应考虑调整大小方法。如果不知道(或不关心)容器的当前大小，那么应该考虑缩小方法。

要使用命令行来缩小 DMS 表空间中一个或多个容器的大小，请输入：

```
ALTER TABLESPACE <name>     REDUCE (FILE '<filename>' <size>)
```

以下示例说明如何在 UNIX 系统的表空间中缩小文件容器(原来为 1000 页)：

```
ALTER TABLESPACE PAYROLL    REDUCE  (FILE '/data/finance'  200)
```

在完成此操作之后，文件大小就从 1000 页减少至 800 页。

要使用命令行来增大 DMS 表空间中一个或多个容器的大小，请输入：

```
 ALTER TABLESPACE <name>          RESIZE (DEVICE '<path>' <size>)
```

以下示例说明如何在 Linux 和 UNIX 系统的表空间中增大两个设备容器(原来大小为 1000 页)：

```
ALTER TABLESPACE HISTORY  RESIZE (DEVICE '/dev/rhd7'  2000,  DEVICE
'/dev/rhd8' 2000)
```

在完成此操作之后，两个设备的大小都从 1000 页增加至 2000 页。

要使用命令行来扩展 DMS 表空间中一个或多个容器的大小，请输入：

```
    ALTER TABLESPACE <name>    EXTEND (FILE '<filename>' <size>)
```

以下示例说明如何在 Windows 系统的表空间中增大文件容器(原来大小为 1000 页)：

```
    ALTER TABLESPACE PERSNEL  EXTEND (FILE '/data1/wrkhist1'  200,
FILE  '/data2/wrkhist2'  200)
```

在完成此操作之后，两个文件的大小都从 1000 页增大至 1200 页。

删除或减少 DMS 容器

对于 DMS 表空间，可以使用 ALTER TABLESPACE 语句从表空间中删除容器或缩小容器的大小。要缩小容器，可以在 ALTER TABLESPACE 语句中使用 REDUCE 或 RESIZE 选项。要删除容器，可以在 ALTER TABLESPACE 语句中使用 DROP 选项。

仅当正在删除或缩小其大小的扩展数据块数目小于或等于表空间中“高水位标记”之上的可用数据块数目时，才允许删除现有表空间容器以及缩小现有容器的大小。高水位标记是表空间中分配的最高页的页数。此标记与表空间中已使用的页的数目不同，高水位标记下的某些扩展数据块可能可供复用。

表空间中高水位标记之上的可用扩展数据块数非常重要，原因是直至高水位标记(包括高水位标记)的所有扩展数据块必须位于表空间内的同一逻辑位置。结果表空间必须有足够的空间才能容纳所有数据。如果没有足够的可用空间，那么会产生一条错误消息(SQL20170N 或 SQLSTATE 57059)。

要删除容器，可在 ALTER TABLESPACE 语句中使用 DROP 选项。例如：

```
    ALTER TABLESPACE TS1 DROP (FILE 'file1', DEVICE '/dev/rdisk1')
```

改变自动存储表空间

对于自动存储表空间，不能手动调整自动存储表空间的大小，数据库管理器将在需要时自动调整容器大小。

3. 重命名表空间

可以使用 RENAME TABLESPACE 语句来重命名表空间。不能重命名 SYSCATSPACE 表空间。不能重命名处于“前滚暂挂”或“正在前滚”状态的表空间。

可以给予现有表空间新名称，而无须关心表空间中的个别对象。重命名表空间时，将更改引用表空间的所有目录记录。例如：

```
    RENAME TABLESPACE TS1 TO TS2
```

注意：

当复原在备份后已被重命名的表空间时，必须在 RESTORE DATABASE 命令中使用新的表空间名。如果使用先前的表空间名，那么将找不到该名称。同样，如果使用 ROLLFORWARD DATABASE 命令前滚表空间，也需要确保使用新名称。如果使用先前的表空间名，那么将找不到该名称。

4. 将表空间从脱机状态切换至联机状态

如果与表空间相关的容器不可访问，这时表空间处于 OFFLINE 状态。要使用命令行从表空间中除去 OFFLINE 状态，请输入：

```
ALTER TABLESPACE <name>  SWITCH ONLINE
```

什么情况下会处于 OFFLINE 状态呢？下面举一个实际生产中的例子。在双机热备 HA 的环境中，客户在主机上重新创建了使用裸设备的表空间后，未同步 HA 环境，结果导致主机故障切换到备机时，由于裸设备权限不正确而导致表空间处于 OFFLINE 状态。

5. 删除表空间

当删除表空间时，也会删除表空间中的所有数据，释放容器，除去目录条目，并导致表空间中定义的所有对象都被删除或标记为无效。可以通过删除表空间来重用空表空间中的容器，但是在试图重用这些容器之前，必须落实 DROP TABLESPACE 语句。

删除用户表空间

可删除包含所有表数据的用户表空间，包括在单个用户表空间中的索引和 LOB 数据。也可删除其中包含的表跨几个表空间的用户表空间。也就是说，可能表数据在一个表空间，索引在另一个表空间且任何 LOB 数据在第 3 个表空间。必须在一条语句中同时删除所有 3 个表空间。包含跨越的表的所有表空间必须全部纳入此单条语句中，否则该删除请求将失败。例如创建表的定义如下：

```
create table xinzhuang pic(picno  int, pic clob(1g)) in data space index in
index_space long in lob_space
```

只能同时删除 3 个表空间：

```
DROP TABLESPACE DATA_SPACE, INDEX_SPACE, LOB_SPACE
```

删除用户临时表空间

仅当用户临时表空间中当前未定义已声明临时表时，才能删除用户临时表空间。当删

除表空间时，不会尝试删除表空间中的所有已声明临时表。

注意：
已声明临时表是在说明应用程序与数据库断开连接时隐式删除的。

删除系统临时表空间

对于默认 4KB 页大小的数据库，如果不首先创建另一系统临时表空间，那么不能删除页大小为 4KB 的默认系统临时表空间。新的系统临时表空间必须具有 4KB 页大小，原因是数据库必须始终存在至少一个具有 4KB 页大小的系统临时表空间。对于默认 8KB、16KB、32KB 页大小的数据库，同样也必须存在一个相应页大小的系统临时表空间。例如，如果具有页大小为 4KB 的单个系统临时表空间，并且想要将一个容器添加到该表空间(为 SMS 表空间)中，那么必须首先添加一个具有适当数目容器的新 4KB 页大小的系统临时表空间，然后删除旧的系统临时表空间(如果正在使用 DMS，那么可以添加容器而不必删除并重新创建表空间)。

默认系统临时表空间的页大小是创建数据库时使用的页大小(默认情况下为 4KB)，但也可以为 8KB、16KB 或 32KB。

下面是用来创建系统临时表空间的语句：

```
    CREATE SYSTEM TEMPORARY TABLESPACE <name>  MANAGED BY SYSTEM USING
('<directories>')
```

创建之后，要使用命令行删除系统表空间，请输入：

```
    DROP TABLESPACE <name>
```

以下 SQL 语句创建名为 TEMPSPACE2 的新的系统临时表空间：

```
    CREATE SYSTEM TEMPORARY TABLESPACE TEMPSPACE2
          MANAGED BY SYSTEM USING ('/data/systemp2')
```

一旦创建 TEMPSPACE2，就可使用以下命令删除原来的系统临时表空间 TEMPSPACE1：

```
    DROP TABLESPACE TEMPSPACE1
```

3.2.3　表空间设计注意事项

1. 表空间类型的选择

在确定应使用哪种类型的表空间来存储数据时，需要考虑一些问题。

SMS 表空间的优点：

- 根据需要，系统按需分配空间
- 由于不必预定义容器，因此创建表空间需要的初始工作较少

DMS 表空间的优点：

- 通过使用 ALTER TABLESPACE 语句，可添加或扩展容器来增加表空间的大小。现有数据可以自动在新的容器集合中重新平衡以保持最佳 I/O 效率
- 根据存储的数据的类型，可以把表长字段(LF)和大对象(LOB)数据、索引和常规表数据分割存放在多个表空间中以提高性能和空间存储容量。通过分隔表数据，可以提高性能和增加每个表存储的数据量。例如，如果要使用 4KB 页大小的大型表空间，那么可以有一个包含 8TB 正规表数据的表、一个包含 8TB 索引数据的单独表空间和另一个包含 8TB 长型数据的单独表空间。如果这 3 种类型的数据存储在一个表空间中，那么总空间将限制为 8TB。使用较大的页大小将允许存储更多数据
- 对范围分区数据创建的索引可以与表数据存储在不同的表空间中
- 可控制数据在磁盘上的位置(如果操作系统允许的话)
- 通常，精心设计的一组 DMS 表空间的性能将优于 SMS 表空间

注意：

对于性能要求很高的应用程序，特别是涉及大量 DML 操作的应用程序，建议使用 DMS 表空间。

其实 DMS 的优势，就是数据在物理磁盘上的连续性。SMS 使用操作系统来管理空间，虽然从逻辑上看，看似所有的文件都是连续的，但是在物理磁盘上，每次文件的增大都必须分配新的空间。所以从操作系统的角度来看，所谓的“分配”不过是在 inode 节点中增加一个指向页的偏移，这个页是操作系统寻找出来没有被使用的，因此从磁盘的角度来看，文件可以被切分成很多块存储在不同的地方——尽管逻辑上它们是连续的。这也就是能够动态增加大小的 SMS 文件的致命伤。不像 DMS，分配完成之后一般不会随意增加或减少大小，SMS 的大小增加有时可能非常频繁，因此每个文件在物理磁盘上的存储会被划分成一个个小块。这样的话，尽管在逻辑上它们的条带化还是连续的，但是从物理磁盘的角度来看，它们的每个 extent 之间可能并非连续，无法使用 range prefetch 直接从磁盘上读取几个连续的 extent。

而且在这两种类型的表空间中，数据的放置也会有所不同。例如，进行高效表扫描要求扩展数据块中的页在物理上是连续的。对于 SMS，操作系统的文件系统决定了每个逻辑

文件页的物理放置位置。根据文件系统中其他活动的级别以及用来确定放置位置的算法的不同，可能会连续分配这些页，也可能不会。但是对于 DMS，因为数据库管理器直接与磁盘打交道，所以可以确保这些页在物理上是连续的。

通常，小型个人数据库用 SMS 表空间管理最容易。另一方面，对于不断增长的大型数据库，建议使用 SMS 表空间用做临时表空间和系统编目表空间，而将具有多个容器的单独的 DMS 表空间用于每个表。另外，建议将长字段(LF)数据和索引存储在它们自己的表空间中。

在深刻理解上述两种表空间的优缺点后，选择表空间时要综合考虑如下因素：

表中的数据量

如果计划在一个表空间中存储许多小表，那么考虑使用 SMS 充当表空间。对于小表，DMS 表现在 I/O 和空间管理效率方面的优点就没有那么重要。SMS 的优点(仅在需要时使用)却对小表更具吸引力。如果表较大或者需要更快地访问表中的数据，应考虑具有较小扩展数据块大小的 DMS 表空间。

设计数据库时，可以考虑对每个非常大的表都使用单独的 DMS 表空间，而将所有的小表组合在单个 SMS 表空间中。这种分隔还允许根据表空间的使用选择适当的扩展数据块大小。

表数据的类型

例如，有的表可能包含不经常使用的历史记录数据；最终用户可能愿意接受较长的响应时间，等待对此数据执行的查询。在这种情况下，可以为历史记录表使用单独的表空间，并将此表空间分配给访问速率较低的较便宜的物理设备。

此外，对于某些表，数据的快速响应时间是非常必要的，需要将这些表分配给快速物理设备的表空间中，这样将有助于支持这些重要的数据需要。如果可以的话，可以使用固态硬盘来存放访问最频繁的表。

通过使用 DMS 表空间，还可以将表数据分发在 3 个不同的表空间中：一个存储索引数据；一个存储大对象(LOB)和长字段(LF)数据；一个存储常规表数据。这允许选择表空间特征和支持最适合该数据的那些表空间的物理设备。例如，可能会将索引数据置于可找到的最快的设备上，这样性能可显著提高。如果将表分布在各个 DMS 表空间中，那么在启用表空间级备份恢复时，应考虑一起备份和复原那些表空间。SMS 表空间不支持以此方式将数据分发在所有表空间中。

管理问题

某些管理功能可以在表空间级执行，但不能在数据库或表级执行。例如，备份表空间

(而不是数据库)可以帮助更好地利用时间和资源，允许频繁地备份带有大量更改的表空间，同时仅偶尔地备份带有少量更改的表空间。

可以复原数据库或表空间。如果不相关的表在同一个表空间中，就可以选择仅复原数据库中较小的部分以降低成本。一种好方法是将相关的表存放在表空间中。这些表可以通过参考约束相关，也可以通过定义的其他业务约束相关。

如果需要经常删除并重新创建特定表，那么应给这样的表单独创建 DMS 表空间，因为删除 DMS 表空间比删除表更有效率。

2. 选择合适的数据页大小

创建表空间时，需要考虑页大小。可以使用 4KB、8KB、16KB 或 32KB 页大小。在选择数据页大小时需要综合考虑空间需求和业务类型(性能需求)以做出选择。

空间需求

因为 DB2 中每个页大小限定了可存储行的最大长度和可存储表空间的最大值，所以选择数据页大小时需要考虑这些。对于 4KB 数据页来说，最多可以存放的行的长度是 4005 字节(4096-91 头部；8KB 为 8192-91；依此类推)，所以首先要根据行的长度来选择数据页大小。表 3-3 列出了每种数据页大小的空间使用限制，以及不同类型的表空间的数据库和索引页大小限制。

表 3-3　表空间特定于页大小的限制

表空间类型	4KB	8KB	16KB	32KB
SMS 表空间	64GB	128GB	256GB	512GB
DMS 表空间(常规)	64GB	128GB	256GB	512GB
DMS 表空间(大型)	8TB	16TB	32TB	64TB
自动存储表空间(常规)	64GB	128GB	256GB	512GB
自动存储表空间(大型)	8TB	16TB	32TB	64TB
临时表空间	64GB	128GB	256GB	512GB

注意：表 3-3 基于 DB2 V9.7

如果数据页大小选择不当，还可能造成空间浪费。例如，如果要使用 32KB 页大小的常规表空间来存储平均大小为 100 字节的行，那么 32KB 的页只能存储 100 * 255 = 25500 字节(24.9 KB)。这意味着每 32KB 中就有大约 7KB 要浪费掉。所以建议创建表空间时，尽

量创建大型表空间，大型表空间的数据页可以存放更多的容量和行数。

业务类型

建议根据业务类型选择合适的数据页大小。通常的业务类型有 OLTP、OLAP、批处理、报表，以及这几种业务的混合类型。下面介绍主要业务类型的特点。

联机事务处理(OLTP)工作负载的特征是：事务需要对数据进行随机访问，通常涉及频繁插入或更新活动和返回一小组数据的查询。通常访问是随机的，并且是访问一页或几页，一般不太可能发生预存取(prefetch)。这里顺便讲一下，其实对于性能要求很高的 OLTP 应用，可以考虑把一些频繁访问的表创建在固态硬盘上。

使用裸设备容器的 DMS 表空间在这种情况下表现得最好。请注意，在 FILE SYSTEM CACHING 关闭的情况下，将 DMS 表空间与文件容器配合使用在某种程度上相当于 DMS 裸设备容器。如果业务逻辑存在大量的随机读，那么 CREATE TABLESPACE 语句中的 EXTENTSIZE 和 PREFETCHSIZE 参数的设置对于 I/O 的效率就显得不是那么重要。

OLAP 查询工作负载的特征是：事务需要对数据进行顺序访问或部分顺序访问，并常常返回大的数据集。使用多个设备容器且每个容器都在单独磁盘上的 DMS 表空间最有可能提供有效的并行预存取。应该将 CREATE TABLESPACE 语句中的 PREFETCHSIZE 参数的值设置为 EXTENTSIZE 参数的值乘以容器设备数之积。此外，可以将预取大小指定为－1，此时数据库管理器将自动(automatic)选择合适的预取大小。这允许数据库管理器以并行方式从所有容器中预取。如果容器的数目发生更改，或者需要使预取更多或更少，那么可以使用 ALTER TABLESPACE 语句相应地更改 PREFETCHSIZE 值；如果把 PREFETCHSIZE 设置为 AUTOMATIC，添加容器后，数据库会自动调整 PREFETCHSIZE 的大小，所以强烈建议把 PREFETCHSIZE 设置为 AUTOMATIC 或－1。

混合工作负载的目标是：对于 OLTP 工作负载，使单个 I/O 请求尽可能有效率；而对于查询工作负载，最大程度地提高并行 I/O 的效率。

选择表空间页大小的注意事项如下所示：

- 对于执行随机行读写操作的 OLTP 应用程序，通常最好使用较小的页大小(4KB、8KB)，这样一来，不需要的行就不会浪费缓冲池空间。
- 对于一次访问大量连续行的决策支持系统(DSS)和 OLAP 应用程序，页大小大一些(16KB、32KB)会比较好，这样就能减少读取特定数目的行所需的 I/O 请求数。较大的页大小还允许减少索引中的层数，因为在一页中可以保留更多的行指针。
- 越大的页，支持的行越长。应根据业务需求选择合适的数据页。
- 在默认的 4KB 页上，表只能有 500 列，而更大的页大小(8KB、16 KB 和 32 KB)支持 1012 列。

- 表空间的最大大小与表空间的页大小成正比，见表 3-3。

3. 扩展数据块在进行大小选择时的注意事项

EXTENTSIZE 指定在跳到下一个容器之前，可以写入容器中的 PAGESIZE 页面的数量。存储数据时数据库管理器反复均衡使用所有容器。该参数只有在表空间中有多个容器时才起作用。选择合理的 EXTENTSIZE 会对表空间的性能产生重大影响。因为这个参数是在创建表空间时定义的，之后不能修改，所以在创建时必须选择合理的 EXTENTSIZE。

表空间的大小和类型

DMS 表空间中的空间一次分配给表一个扩展数据块。当插入该表而一个扩展数据块变满时，会分配新的扩展数据块，直到彻底用完容器为止。

将 SMS 表空间中的空间一次分配给表一个扩展数据块或者一次分配给表一页。当插入该表而一个扩展数据块或页变满时，会分配新的扩展数据块或页，直到使用了文件系统中的所有扩展数据块或页为止。当使用 SMS 表空间时，允许进行多页文件分配(DB2 V8 之前需要执行 db2empfa 以激活多页分配，V8 以后默认自动设置了多页分配)。多页文件分配允许分配扩展数据块而不是一次分配一页。

每个表对象都是单独存储的，每个对象按需要分配新的扩展数据块。每个 DMS 表对象还与称为扩展数据块映像的元数据对象配成一对，该元数据对象描述表空间中属于表对象的所有扩展数据块。用于扩展数据块映像的空间也是以一次一个扩展数据块的方式分配。因此，DMS 表空间中对象的初始空间分配是两个扩展数据块(SMS 表空间中对象的初始空间分配是一页)。

如果在 DMS 表空间中有多个较小的表，那么可能要分配相对大的空间来存储相对少量的数据。在这种情况下，应该指定小的扩展数据块大小。另一方面，如果有一个增长速率高的非常大的表，并且使用具有较小扩展数据块大小的 DMS 表空间，那么可能会产生与其他扩展数据块的频繁分配相关的不需要的开销。

下面的经验法则建立在表空间中每个表的平均大小的基础上：

- 如果小于 50 MB，EXTENTSIZE 为 8
- 如果介于 50MB 到 500MB 之间，EXTENTSIZE 为 16
- 如果介于 500 MB 到 5GB 之间，EXTENTSIZE 为 32
- 如果大于 5GB，EXTENTSIZE 为 64

针对这些表的访问类型

OLAP 数据库和大部分对表的访问包括许多查询或处理大量数据的事务(仅限于查询)的表，或者增长速度很快的表，从表中预取数据可以显著改善性能，应使用较大的 extent。

反之，对于较小的频繁更改、频繁随机读取的小的配置表，建议使用较小的 extent。

如果打算在表的设计中使用多维聚簇(MDC)，扩展数据块就是重要的设计决定之一。MDC 表将为创建的每个新的维集分配扩展数据块。如果扩展数据块太大，那么扩展数据块的很大一部分有可能是空的(对于包含很少记录的维集)，这会造成非常大的空间浪费。关于 MDC 及其对 EXTENTSIZE 影响的更多信息，请参见《高级进阶 DB2(第 2 版)》一书。

3.2.4　prefechsize 大小选择

为了提高数据库缓冲池的命中率，数据库通过预取操作在查询使用所需的数据之前读入这些数据，因为数据已经存在于内存中了，这样一来，查询在使用这些数据的时候就不必等待执行 I/O 了。当数据库管理器确定顺序 I/O 是适当的，并且确定预取操作可能有助于提高性能时，就选择预取操作。

通过使用 ALTER TABLESPACE 可以轻易地修改预取大小。最优设置差不多是下面这样的：

```
    Prefetch Size = (# Containers of the table space on different physical disks)
* Extent Size
```

如果表空间驻留在某个磁盘阵列上，那么进行如下设置：

```
    PREFETCH SIZE = EXTENT SIZE * (# of non-parity disks in array)
```

注意：
在 DB2 V9 版本以后，可以在创建表空间的时候自动预取大小。

在添加或删除容器后，可能忘记更新表空间的预取大小，此时应考虑允许数据库管理器自动确定预取大小。如果忘记更新预取大小，那么数据库性能可能会明显降低。所以可以在创建表空间时指定 PREFETCHSIZE 为 AUTOMATIC，这样就可以设置自动预取大小，并可以通过下面的快照监控来查看是否设置自动预取：

```
    db2 get snapshot for tablespaces on sample | more
              表空间快照
    第一个数据库连接时间戳记              = 2012-10-15 09:19:37.992116
    ...........................................................
     表空间扩展数据块大小(以页计)        = 4
      启用的自动预取大小 (prefetchsize)=是(automatic)
      当前正在使用的缓冲池标识              = 1
      下一次启动的缓冲池标识            = 1
      使用自动存储                      =是
      启用自动调整大小                  =是
```

```
    文件系统高速缓存                      = 否
    表空间状态                            = 0x'00000000'
      详细解释:        正常
    表空间预取大小(以页计)                = 4
  略.......................
```

当然，PREFETCHSIZE 的大小设置还和 EXTENTSIZE 的设置有关，所以首先要合理地设置 EXTENTSIZE 的大小，然后再根据 EXTENTSIZE 的大小设置 PREFETCHSIZE。比较好的建议是创建数据库时采用自动存储。这样可由数据库管理器自动设置 EXTENTSIZE 和 PREFETCHSIZE 的大小。

3.2.5 文件系统(CIO/DIO)和裸设备

在创建 DMS 表空间容器时可以选择使用裸设备或文件系统，下面来看看两者的区别。我们知道，内存的读写效率比磁盘高近万倍，因此数据库通常会在内存中开辟一片区域，称为 Buffer Pool，使数据的读写尽量在这部分内存中完成。同样，在文件系统中，操作系统为了提高读写效率，也会为文件系统开辟一块 Buffer 用于读写数据的缓存。这样，数据库中的数据会被缓存两次。为了避免操作系统的这次缓存，可以采用裸设备作为数据文件的存储设备。裸设备也称为裸分区(Raw Partition)，是没有被加载(mount)到操作系统的文件系统中的磁盘分区，通过字符设备驱动来访问。裸设备的 I/O 读写不由操作系统控制，而是由应用程序(如数据库)直接控制。

裸设备的优点：

- 由于屏蔽了文件系统缓冲区而进行直接读写，因此具有更好的性能。对硬盘的直接读写就意味着取消了硬盘与文件系统的同步需求。这一点对于纯 OLTP 系统非常有用，因为在这种系统中，读写的随机性非常大，以至于一旦数据被读写之后，它们在今后较长的一段时间内不会得到再次使用。除了 OLTP，裸设备还能够从以下几个方面改善 DSS(决策支持系统)应用程序的性能：
  - ◊ 排序：对于 DSS 环境中大量存在的排序需求，裸设备提供的直接写功能也非常有用，因为对临时表空间的写动作速度更快。
  - ◊ 顺序访问：裸设备非常适合于顺序 I/O 动作。同样，DSS 中常见的顺序 I/O(表/索引的全表扫描)使得裸设备更加适用于这种应用程序。
- 直接读写，不需要经过操作系统级的缓存。节约了内存资源，在一定程度上避免了内存的竞争。
- 避免了操作系统的 cache 预读功能，减少了 I/O。
- 采用裸设备避免了文件系统的开销，比如维护 i-node、空闲块等。

裸设备的缺点：

- 裸设备的空间大小管理不灵活。在放置裸设备的时候，需要预先规划好裸设备的空间使用。还应当保留一部分裸设备以应付突发情况，这也是对空间的浪费。
- 需要操作系统 root 用户干预，因为裸设备的创建、更改权限、扩展大小等都需要由 root 用户完成，这增加了管理的成本。

文件系统的优点：

文件系统易于管理和维护，比如文件的基本管理以及安全和备份等。

文件系统的缺点：

性能比不上裸设备。

我们在选择表空间容器时，从性能上考虑尽量采用裸设备，但是如果使用自动存储方式创建数据库和表空间，这种方式不支持裸设备。或者为了便于管理而采用文件系统方式，这时候需要合理设置文件系统相关选项和表空间相关选项。下面讲解文件系统方面应该注意的事项。

CIO/DIO

直接 I/O(DIO)由于可以绕过在文件系统级别进行高速缓存，因此能改进内存性能。此过程可减少 CPU 开销并使得更多的内存可用于数据库实例。并发 I/O(CIO)具有 DIO 的优点，并且还可以消除串行化写访问权。与使用文件系统缓冲 I/O 相比，在具有大量事务处理工作负载和回滚时，CIO/DIO 机制可增大吞吐量。

DIO 和 CIO 在 HP-UX、Solaris、Linux 和 Windows 操作系统的最新版本中都受支持，具体的支持列表请查看最新的 DB2 信息中心。

关键字 NO FILE SYSTEM CACHING 和 FILE SYSTEM CACHING 是 CREATE 和 ALTER TABLESPACESQL 语句的一部分，允许指定将对每个表空间使用 DIO 还是 CIO。当 NO FILE SYSTEM CACHING 有效时，只要可能，数据库管理器都会尝试使用“并发 I/O”。在不支持 CIO 的情况下(例如，当使用了 JFS 时)，将取而代之使用 DIO。

建议在表空间级别启用或禁用 UNIX、Linux 和 Windows 中的非缓冲 I/O。这将允许在特定表空间上启用或禁用非缓冲 I/O，同时避免数据库物理布局中的任何依赖性。另外还允许数据库管理器确定每个文件最适合使用哪种 I/O，缓冲的还是非缓冲的。

NO FILE SYSTEM CACHING 子句用于启用非缓冲 I/O，从而禁用特定表空间的文件高速缓存。一旦启用非缓冲 I/O，数据库管理器就会根据平台自动确定将使用直接 I/O 还是并发 I/O。由于使用 CIO 可以提高性能，因此 CIO 只要受支持，数据库管理器就会使用。

FILE SYSTEM CACHING 选项并不是总没有好处，例如当应用程序检索 LOB 或 LONG 数据时，这些大对象数据不能经过数据库缓冲池，每次应用程序需要其中一个页时，数据

库管理器必须从磁盘对之进行直接读取。但是，如果 LOB 或 LONG 数据存储在 SMS 或 DMS 文件容器中，文件系统的高速缓存可提供缓冲，因此也就改善了性能。

未在 CREATE TABLESPACE 语句或 CREATE DATABASE 命令中指定此属性时，数据库管理器将使用基于平台和文件系统类型的默认行为处理请求。为了查看是否启用 FILE SYSTEM CACHING 属性，可以使用：

- GET SNAPSHOT FOR TABLESPACES 命令(此命令会在第 12 章讲解)

例如，以下是 DB2 GET SNAPSHOT FOR TABLEPSACES ON SAPMPLE 命令的输出：

```
表空间名                                    = USERSPACE1
   表标识                                   = 2
   表空间类型                               =数据库管理的空间
   表空间内容类型                           = 所有永久数据。大型表空间。
   表空间页大小(以字节计)                   = 4096
   表空间扩展数据块大小(以页计)             = 32
    已启用自动预取大小                      = Yes
    当前正在使用的缓冲池标识                = 1
    下一次启动的缓冲池标识                  = 1
    使用自动存储                            = Yes
    已启用自动调整大小                      = Yes
   文件系统高速缓存                         = No
   表空间状态                               = 0x'00000000'
      ..........................略.......................
```

- db2pd -d <dbname> -tablespaces 命令(此命令会在第 12 章讲解)
- db2look -d <dbname> l 命令(此命令会在第 13 章讲解)

下面举几个关于文件缓存的例子。

例 3-1 假定数据库和所有相关表空间容器位于 AIX JFS 文件系统中，并且发出了以下语句：

```
DB2 "CREATE TABLESPACE DATA SPACE MANAGED BY DATABASE USING (file '/data/db2data'
800M)"
```

在先前版本中，如果未指定该属性，那么数据库管理器将使用缓冲 I/O(FILE SYSTEM CACHING)作为 I/O 机制；对于 DB2 V9.5，数据库管理器使用 NO FILE SYSTEM CACHING。

例 3-2 在以下语句中，NO FILE SYSTEM CACHING 子句指示对于此特定表空间，文件系统的高速缓存将 OFF：

```
CREATE TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

例 3-3 以下语句对现有表空间禁用文件系统的高速缓存：

```
ALTER TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

例 3-4 以下语句对现有表空间启用文件系统的高速缓存：

```
ALTER TABLESPACE table space name ... FILE SYSTEM CACHING
```

经过上面的讲解，建议创建表空间时，表空间的容器采用裸设备或支持并发 I/O 或直接 I/O 的文件系统。

3.2.6 设置 OVERHEAD 和 TRANSFERRATE

这两个参数用于确定查询优化期间的 I/O 成本。这两个值的测量单位都是毫秒，而且它们应当分别是所有容器开销和传送速率的平均值。开销是与 I/O 控制器活动、磁盘寻道时间和旋转延迟时间相关联的时间。传送速率是将一页读入内存所必需的时间量。它们的默认值分别是 24.1 和 0.9。可以根据硬件规格计算这些值。

```
Transrate = (1/传送速率)*1000/1024000*4096(假设用 4KB 页大小)
Overhead = 平均寻道时间+(((1/磁盘转速)*60*1000)/2)
```

而平均寻道时间、磁盘旋转速度和传送速率是由硬盘本身决定的(可以使用操作系统命令或从硬盘厂商获得底层硬盘的物理特性)。

所以我们必须合理地设置这两个值，以便让优化器了解底层存储的物理特性，从而制订最优的执行计划。

3.2.7 优化 RAID 设备上表空间的性能

现在很多应用系统都把数据库存放在“独立磁盘冗余阵列”(RAID)设备上，要优化存放在 RAID 设备上的表空间性能，请遵循下列准则：

- 在一组 RAID 设备上创建表空间时，应该把表空间容器创建在多个 RAID GROUP 上。
- 考虑以下示例：将 15 个 146GB 磁盘配置为 3 个 RAID-5 阵列，每个阵列包含 5 个磁盘。格式化以后，每个磁盘可容纳大约 136GB 数据。因此，每个阵列可存储大约 544GB(4 个活动磁盘×136GB)数据。如果表空间需要 300GB 存储空间，那么创建 3 个容器并将每个容器分别放在 3 个不同的 RAID 设备上。每个容器使用设备上的 100GB(300GB/3)，并且每个设备上保留 444GB(544GB−100GB)以提供附加表空间。

- 为表空间选择适当的扩展数据块(extent)大小。理想状态下，扩展数据块大小应该等于磁盘底层 stripe 大小或其倍数，其中 stripe 的大小等于 strip 的大小乘以活动磁盘数，strip 大小表示磁盘控制器在向一个物理磁盘写入多少数据后才转向下一个物理磁盘。这样可以确保基于扩展数据块的操作(比如预取时的并行顺序读取)不会争用相同的物理磁盘。
- 在上面我们举的那个示例中，如果 strip 大小为 64KB，而页大小为 16KB，那么适当的扩展数据块大小可能是 256KB(64KB*4)。
- 使用 DB2_PARALLEL_IO 注册变量为所有表空间启用并行 I/O，并为每个容器指定物理磁盘数。在设置个变量之前，我们先了解这个变量的含义。

DB2_PARALLEL_IO 注册变量用来确定每个容器的底层物理硬盘数以及对表空间上并行 I/O 的影响。当为表空间设置多个容器或者设置 DB2_PARALLEL_IO 注册变量时，会为表空间启动并行预取。并行预取请求的个数是由表空间容器个数和 DB2_PARALLEL_ IO 同时决定的，每个预取请求都只能读取一个 EXTENT。如果未设置 DB2_PARALLEL_IO，那么表空间的并行度与容器数目相等。否则，表空间的并行度由表空间预取大小和 EXTENT 大小决定，建议大家将预取大小设置为 AUTOMATIC，这样就由 DB2 自动计算预取大小，让 DB2 选择最合适的并行度。

DB2_PARALLEL_IO=TablespaceID:[*n*]，如果没有指定 *n*，那么使用默认值 6(就是假定容器跨越 6 个 RAID 底层物理磁盘)。以下是 DB2_PARALLEL_IO 注册变量如何影响预取大小的若干示例(假设已使用 AUTOMATIC 预取大小定义以下所有表空间)。

- DB2_PARALLEL_IO=*

“*”表示所有表空间均会启用并行 I/O，由于没有指定 *n*，所有表空间将使用每个容器磁盘数目等于 6 时的默认值。将预取请求分解成“6×容器数目”个并行请求，每个请求的读取大小为 extent 大小。

- DB2_PARALLEL_IO=*:3

“*”表示所有表空间均会启用并行 I/O。“*:3”表示所有表空间将 3 作为每个容器的默认磁盘数目。将预取请求分解成“3×容器数目”个并行请求，每个请求的读取大小为 extent 大小。

- DB2_PARALLEL_IO=*:3，1:1

“*”表示所有表空间均会启用并行 I/O。“*:3”表示所有表空间将 3 作为每个容器的磁盘个数，将预取请求分解成“3×容器数目”个并行请求，每个请求的读取大小为 extent 大小。对于 ID 为 1 的表空间，将预取请求分解成“1×容器数目”个并行请求，每个请求的读取大小为 extent 大小。

对于此例中的情况，请将 DB2_PARALLEL_IO 设置为*:4，如图 3-17 所示。

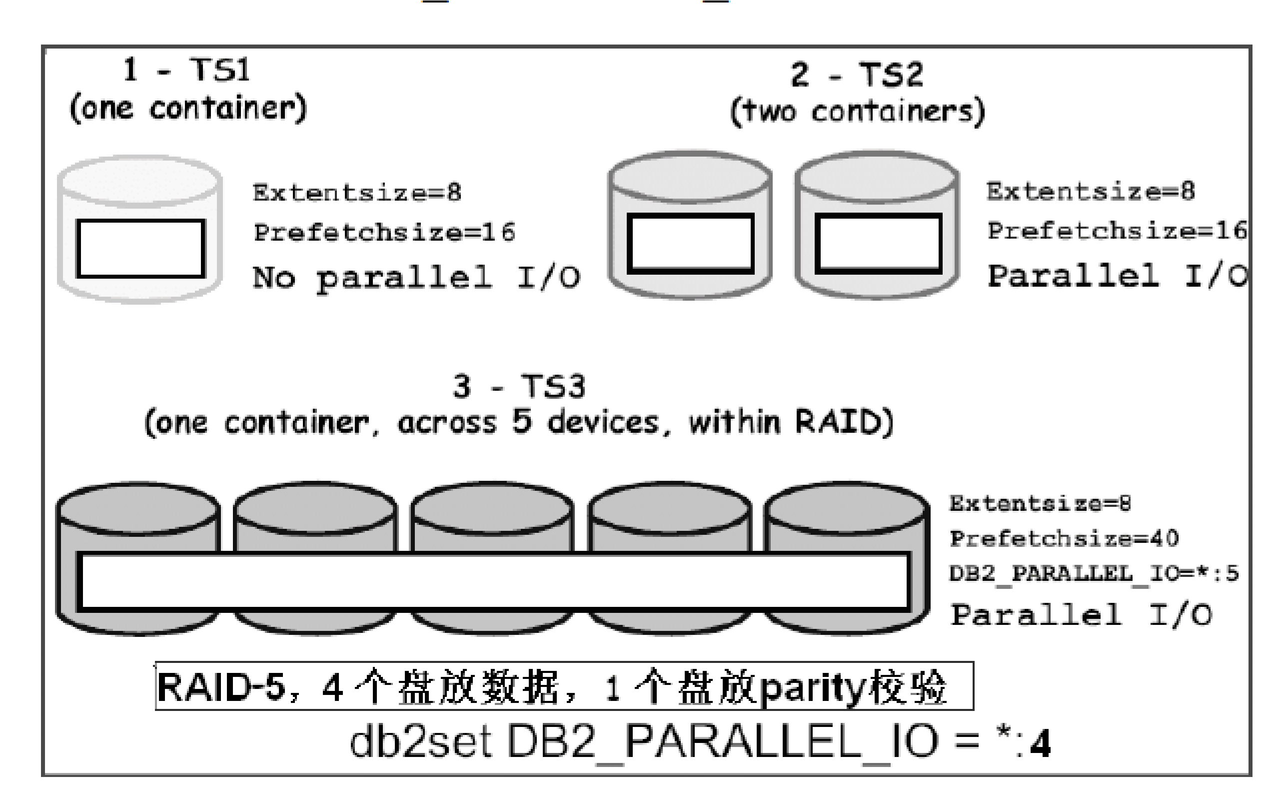


图 3-17　将 DB2_PARALLEL_IO 设置为*:4

如果将表空间的预取大小设置为 AUTOMATIC，那么数据库管理器将使用为 DB2_PARALLEL_IO 指定的物理磁盘数值来确定预取大小值。如果预取大小未设置为 AUTOMATIC，那么可以手动设置此值，请考虑 RAID 条带大小，它是 strip 大小乘以活动磁盘数产生的值。考虑满足下列条目的预取大小值：

- 等于 RAID 条带大小乘以 RAID 并行设备数(或此乘积的整数表示)。
- 是扩展数据块大小的整数表示。

在上面示例中，可将预取大小设置为 768KB。此值等于 RAID 条带大小(256KB)乘以表空间的 RAID 并行容器设备数(3)，也是扩展数据块大小(256KB)的倍数。选择此预取大小意味着单个预取会涉及所有阵列中的所有磁盘。如果因为工作负载主要涉及大量扫描而希望预取程序更积极地工作，那么可改为使用此值的倍数，如 1536 KB(768KB×2)。

不要设置 DB2_USE_PAGE_CONTAINER_TAG 注册变量。如之前所述，应使用等于 RAID 条带大小或其倍数的扩展数据块大小来创建表空间。但是，将 DB2_USE_PAGE_CONTAINER_TAG 设置为 ON 时，将使用单页容器标记，并且扩展数据块不会与 RAID 条带对齐。因此，在 I/O 请求期间可能需要访问比最优情况更多的物理磁盘。

3.2.8　合理设置系统临时表空间

系统临时表空间主要用于分组、排序、连接、重组和创建索引等。要确保系统临时表空间的最大页大小对于查询或定位更新来说足够大。

DB2 V9 中大记录标识符(RID)的使用增加了来自查询或定位更新的结果集的行大小。如果结果集中的行大小接近于现有系统临时表空间的最大行长度限制，那么可能需要创建

具有更大页大小的系统临时表空间。下面举个例子：

假如表 T1 具有 20 个字段，T2 具有 18 个字段，每行最大长度分别为 3500 字节和 3000 字节，它们能正常地存放在 4KB 表空间中。但是如果发出如下这条 SQL 语句：

```
select T1.*,T2.* from T1,T2 where T1.id=T2.id
```

对于上面这条 SQL 语句，在临时表空间中一行的长度已经达到 6500(3500+3000)字节大小，这时原来 4KB 的临时表空间已经不能存放，必须使用更大页大小的临时表空间。

所以要确保系统临时表空间的最大页大小对于查询或定位更新足够大，否则会显著影响性能。可以使用如下方法：

- 确定来自查询或定位更新的结果集的最大行大小。使用曾用来创建表的 DDL 语句监控查询或者计算最大行大小。
- 检查结果集中的最大行大小是否适合系统临时表空间的页大小：

```
maximum_row_size>maximum_row_length-8 字节(单分区结构开销)
```

其中，maximum_row_size 是结果集的最大行大小，maximum_row_length 是基于所有系统临时表空间的最大页大小所允许的最大长度，应根据表空间页大小确定最大行长度。

- 创建系统临时表空间，其大小应至少比创建了表的表空间页大小大一页大小(假设还没有这样大小的系统临时表)。例如，在 UNIX 操作系统中，如果在具有 4KB 页大小的表空间中创建了表，那么使用 8KB 页大小创建额外系统临时表空间以备需要的时候使用：

```
CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp PAGESIZE 8K
    MANAGED BY SYSTEM  USING ('/data1/tmp_tbsp','/data2/tmp_tbsp')
```

如果表空间页大小是 32KB，那么可以减少在查询中选择的信息或者分开这些查询以适合系统临时表空间页。例如，如果选择了表的所有列，那么可以改为仅选择真正需要的列或者选择某些列的子串来避免超出页大小限制。

3.3 缓冲池

缓冲池指的是从磁盘读取表和索引数据时，数据库管理器分配的用于高速缓存这些表或索引数据的内存区域。每个 DB2 数据库都必须具有至少一个缓冲池。数据库中的数据访问都需要经过缓冲池：读的数据需要先读到缓冲池才能提交给应用，写的数据也是要先写到缓冲池才能进行 I/O。缓冲池是影响数据库性能的最大参数，所以必须合理地设计缓

冲池。

创建数据库时，DB2 会自动地创建名为 IBMDEFAULTBP 的默认缓冲池，所有的表空间都共享该缓冲池。可以使用 CREATE BUFFERPOOL、DROP BUFFERPOOL 和 ALTER BUFFERPOOL 语句来创建、删除和修改缓冲池。SYSCAT.BUFFERPOOLS 目录视图记录数据库中定义的缓冲池的信息。从 DB2 V9 开始，缓冲池的默认大小都是自动调整，如果想取消自动调整，可以通过在 CREATE BUFFERPOOL 命令中指定 SIZE 关键字来设置固定值。足够的缓冲池大小是数据库拥有良好性能的关键所在，因为这可以减少磁盘 I/O 这一最耗时操作。大型缓冲池还会对查询优化产生影响，因为更多的工作可在内存中完成，而无须进行 I/O。

3.3.1　缓冲池的使用方法

首次访问表中的数据行时，数据库管理器会将包含数据的页放入缓冲池中。这些页将一直保留在缓冲池中，直到关闭数据库或者其他页需要使用某一页占用的空间为止。缓冲池中的页可能正在使用，也可能没有使用，它们可能是脏页，也可能是干净页。

- 正在使用的页就是当前正在读取或更新的页。为了保持数据的一致性，数据库管理器只允许一次只有一个代理程序更新缓冲池中的给定页。如果正在更新某页，那么只能允许一个代理程序互斥地访问。如果正在读取页，那么多个代理程序可以同时读取该页。
- “脏”页包含已更改但尚未写入磁盘的数据。
- 将已更改的页写入磁盘之后，该页就是“干净”页，并且可能仍然保留在缓冲池中。

大多数情况下，调整数据库涉及设置用于控制将数据移入缓冲池以及等待将数据从缓冲池写入磁盘的配置参数。如果最近的代理程序不需要页空间，那么可以将页空间用于新应用程序中的新页请求。额外的磁盘 I/O 会使数据库管理器性能下降。

可使用数据库监控工具来计算缓冲池的命中率，缓冲池的命中率可帮助你调整缓冲池。这部分内容我们会在第 9 章讲解。

3.3.2　缓冲池和表空间之间的关系

设计缓冲池时，需要了解表空间与缓冲池之间的关系。每个表空间都与特定的缓冲池相关。IBMDEFAULTBP 是默认缓冲池。数据库管理器还会分配下列系统缓冲池：IBMSYSTEMBP4K、IBMSYSTEMBP8K、IBMSYSTEMBP16K 和 IBMSYSTEMBP32K(以前称为“隐藏缓冲池”)。要使另一个缓冲池与表空间相关，该缓冲池必须存在并且它们具有相同的页大小。关联是在使用 CREATE TABLESPACE 语句创建表空间时定义的，但以后可使用 ALTER TABLESPACE 语句更改关联。

如果拥有多个缓冲池，那么可以配置数据库使用更多的内存以改善整体性能。例如，对于 OLAP 类型的应用，我们建议采用大的缓冲池以利于大块顺序读取；用于联机事务应用程序的表空间可以根据业务特点使用多个小的缓冲池，以便可以更长时间地高速缓存应用程序使用的数据页，使响应时间更快。

图 3-18 是一个表空间和缓冲池设计示例。

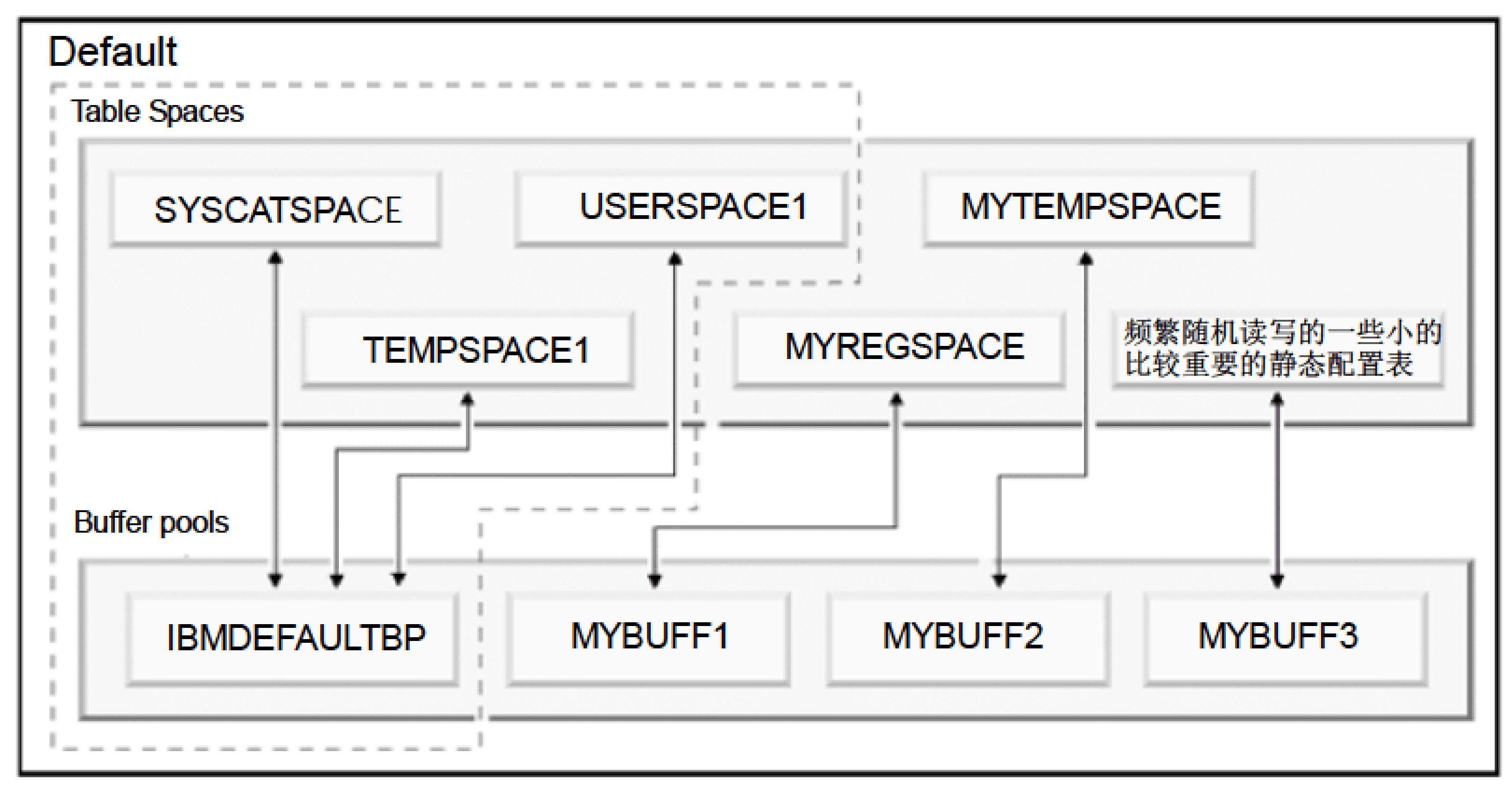


图 3-18　表空间和缓冲池之间关系的设计示例

该数据库有 6 个表空间：数据库创建时默认生成的 3 个表空间——系统编目表空间、系统临时表空间和 USERSPACE1(使用默认的缓冲池 IBMDEFAULTBP)；用户定义的常规表空间 MYREGSPACE(使用 MYBUFF1 缓冲池)、MYTEMPSPACE(使用 MYBUFF2 缓冲池)；最后一个表空间用于放置一些频繁随机读写的比较重要的静态配置表(使用 MYBUFF3 缓冲池)。在图 3-18 中，我们没有看到为 LONG 表空间设置的缓冲池，这是因为大对象的读取不能经过内存，从磁盘直接读取。

3.3.3　维护缓冲池

缓冲池的页大小

默认缓冲池的页大小是在使用 CREATE DATABASE 命令时设置的，此默认值表示所有将来 CREATE BUFFERPOOL 和 CREATE TABLESPACE 语句的默认页大小。如果在创建数据库时不指定页大小，那么默认页大小是 4KB。

注意：

如果确定数据库需要 8KB、16 KB 或 32 KB 的页大小，那么必须至少定义一个具有相匹配的页大小并且与数据库中表空间相关联的缓冲池。选择用于缓冲池的页大小是很重要

的，这是因为创建缓冲池之后就不能改变页大小了。

基于块(block)的缓冲池

DB2 允许留出缓冲池的一部分(最高可达 98%)用于基于块的预取操作。基于块的 I/O 可以通过将块读入相邻的内存区而不是将块分散装入单独的页，从而提高预取操作的效率。每个缓冲池的块大小必须相同，并且由 BLOCKSIZE 参数进行控制。该值等于块的大小(单位为页)，取值范围从 2 到 256，默认值为 32。

注意：
基于块的缓冲池主要用于数据仓库、DSS 之类的连续大块读写的应用中。

在创建新的缓冲池之前，应解决下列问题：

- 想要使用什么缓冲池名称？
- 是立即创建缓冲池，还是在下一次取消激活，然后重新激活数据库之后创建缓冲池？
- 希望缓冲池的页大小是多大？
- 是将缓冲池设为固定大小，还是由数据库管理器自动调整缓冲池大小以对工作负载做出响应？建议在创建缓冲池期间不指定 SIZE 参数，从而允许数据库管理器自动调整缓冲池。
- 是否想保留一部分缓冲池用于基于块的 I/O？
- 设计缓冲池时，还应根据机器上已安装的内存量以及与数据库管理器在同一机器上同时运行的其他应用程序所需要的内存来考虑内存要求。当没有足够内存来保存所访问的所有数据时，操作系统就会进行数据交换。将某些数据写入或交换到临时磁盘存储器中以为其他数据腾出空间时，就会进行数据交换。当需要临时磁盘存储器上的数据时，又会将数据交换回内存中。

创建缓冲池

恰当地定义缓冲池是拥有运行良好的系统的关键之一。对于 32 位操作系统，知道内存的寻址空间十分重要(AIX 是 1.75 GB；Linux 是 1.75 GB；Sun Solaris 是 3.35 GB；HP-UX 是大约 800MB；Windows 是 2GB~3GB)。64 位系统没有这样的界限。

使用 CREATE BUFFERPOOL 语句定义数据库管理器要使用的新缓冲池。以下是基本 CREATE BUFFERPOOL 语句的示例：

```
CREATE BUFFERPOOL BP3 SIZE 2000 PAGESIZE 8K
```

创建缓冲池的两个关键参数是 IMMEDIATE 和 DEFERRED。当使用 IMMEDIATE 参数时，将立即更改缓冲池大小，而不必等到下一次激活数据库时才生效。默认情况下，新的缓冲池是使用 IMMEDIATE 关键字创建的。对于立即请求，不需要重新启动数据库，将立即激活缓冲池。如果数据库共享内存不足以分配新空间，那么会延迟(DEFERRED)运行该语句。

如果发出 CREATE BUFFERPOOL DEFERRED，那么不会立即激活缓冲池；将在下一次启动数据库时创建缓冲池。在重新启动数据库之前，任何新的表空间都将使用现有缓冲池，即使创建表空间时显式使用延迟缓冲池也是如此。

创建缓冲池时，应查看机器上是否有足够的内存用于已创建的所有缓冲池。要综合考虑操作系统中别的应用和操作系统本身的内存需求。

修改缓冲池

有许多理由要修改缓冲池，例如，为了启用自调整内存功能。为此，可以使用 ALTER BUFFERPOOL 语句。可以修改缓冲池的如下属性：

- 启用缓冲池自调整功能，从而允许数据库管理器根据工作负载调整缓冲池大小。
- 修改基于块的 I/O 的缓冲池的块区域。
- 修改部分缓冲池的大小。

使用 ALTER BUFFERPOOL 语句改变缓冲池对象的单个属性。例如：

```
ALTER BUFFERPOOL buffer pool name SIZE number of pages
```

buffer pool name 是缓冲池名称，number of pages 是要分配给特定缓冲池的新页数。也可以使用值－1，用于指示缓冲池大小应该是在 BUFFPAGE 数据库配置参数中设置的值。

查看缓冲池属性

通过查询 SYSCAT.BUFFERPOOLS 系统视图可以列出缓冲池信息：

```
SELECT * FROM SYSCAT.BUFFERPOOLS
BPNAME          BUFFERPOOLID NGNAME             NPAGES      PAGESIZE    ES
--------------- ------------ ------------------ ----------- ----------- --
IBMDEFAULTBP               1 -                          250        4096 N
  1 record(s) selected.
```

要找出哪个缓冲池被分配给了表空间，请运行下面这个查询：

```
SELECT TBSPACE, BUFFERPOOLID FROM SYSCAT.TABLESPACES
TBSPACE          BUFFERPOOLID
```

```
------------------ ------------
SYSCATSPACE                  1
TEMPSPACE1                   1
USERSPACE1                   1
  3 record(s) selected.
```

可以在上一个查询中找到 BUFFERPOOLID，该查询使你能够看到每个表空间与哪个缓冲池相关联。

删除缓冲池

删除缓冲池时，应确保没有任何表空间已指定给这些缓冲池。不能删除 IBMDEFAULTBP 缓冲池。

可以使用 DROP BUFFERPOOL 语句删除缓冲池，如下所示：

```
DROP BUFFERPOOL <buffer pool name>
```

3.3.4　缓冲池的设计原则

缓冲池的命中率

使用多个用户表空间的最重要原因是管理缓冲池的命中率。一个表空间只能与一个缓冲池相关联，而一个缓冲池则可用于多个表空间。

缓冲池调优的目标是帮助 DB2 尽可能好地利用可用于缓冲池的内存。整个缓冲池大小对 DB2 性能有巨大影响，这是因为缓存大量的页可以显著地减少 I/O 这一最耗时操作。但是，如果总的缓冲池设置太大，并且没有足够的物理内存来分配给它们，那么系统将会使用隐藏缓冲池，性能就会急剧下降。要计算最大的缓冲池大小，需要综合考虑 DB2、操作系统以及其他任何应用程序内存的使用率。一旦确定 DB2 总的可用内存大小，就可以将这个区域划分成不同的缓冲池以提高命中率。如果有一些具有不同页大小的表空间，那么每种页大小必须至少有一个缓冲池。

拥有多个缓冲池可以最大限度地将数据保存在缓冲池中。例如，假设数据库有许多频繁使用的小型表，这些表通常都位于缓冲池中，因此访问起来就非常快。现在假设有针对非常大的表运行的查询，使用同一缓冲池并且需要读取比总的缓存池大小还多的页。当查询运行时，之前来自这些频繁使用的小型表的页将会丢失，这使得再次需要这些数据时就必须重新读取它们。

如果小型表拥有自己的缓冲池，那么它们就必须拥有自己的表空间，在这种情况下其他的查询就不能覆盖它们的页。这有可能产生更好的整体系统性能，虽然这会对大型查询

造成一些小的负面影响。经常性地进行调优是为了实现整体性能的提高，而且时常需要在不同的系统功能之间做出权衡。区分功能的优先级并记住总吞吐量和使用情况，同时对系统性能进行调整，这是非常重要的。

DB2 能够在不关闭数据库的情况下更改缓冲池大小。带有 IMMEDIATE 选项的 ALTER BUFFERPOOL 语句会立刻生效，只要数据库共享的内存中有足够的保留空间可以分配给新空间。可以使用这个功能，根据使用过程中的周期变化(例如从白天的交互式使用转换到夜间的批处理工作)来调优数据库性能。

关于如何监控和调整缓冲池，请详细参考本书“第 9 章：DB2 基本监控方法”。

确定有多少缓冲池

对于由数据库中表空间使用的每一种页面大小，都需要至少一个缓冲池。通常，默认的 IBMDEFAULTBP 缓冲池是留给系统编目的。为处理表空间的不同页面大小和行为，需要创建新的缓冲池。

建议为每种页面大小使用缓冲池，对于 OLAP/DSS 类型的工作负载更是如此。DB2 在缓冲池的自我调优方面十分擅长，并且会将经常被访问的行放入内存，因此多数情况下对于每一种页的大小创建一个缓冲池就足够了(这一选择也避免了管理多个缓冲池的复杂性)。

如果时间允许，并且需要进行改进，那么可能希望使用多个缓冲池。其思想是将访问最频繁的行放入缓冲池中。在那些随机访问或者很少访问的表之间共享缓冲池可能会给缓冲池带来“污染”，因为有时候要为本来可能不会再去访问的行消耗空间，甚至可能将经常访问的行挤出到磁盘上。如果将索引保留在它们自己的缓冲池中，那么在索引使用频繁的时候(例如索引扫描)，还可以显著地提高性能。

这与我们对表空间的讨论是紧密联系的，因为要根据表空间中表的行为来分配缓冲池。如果采用多缓冲池的方法，对于初学者来说使用 4 个缓冲池比较合适：

- 一个中等大小的缓冲池，用于临时表空间。
- 一个大型的缓冲池，用于索引表空间。
- 一个大型的缓冲池，用于那些包含经常要访问的表的表空间。
- 一个小型的缓冲池，用于那些包含访问不多的表、随机访问的表或顺序访问的表的表空间。

对于只包含 LOB 数据的 DMS 表空间，可以为其分配任何缓冲池，因为 LOB 不占用缓冲池空间。

确定为缓冲池分配的内存

千万不要为缓冲池分配多于所能提供的内存，否则就会招致代价不菲的操作系统内存分页(memory paging)。通常来讲，如果没有进行监控，要想知道一开始为每个缓冲池分配多少内存是十分困难的。

对于 OLTP 类型的工作负载，一开始将 25%(仅为参考，实际大小请参考自己操作系统中的内存资源和运行在操作系统中的应用情况)的可用内存分配给缓冲池比较合适。

对于 OLAP/DSS，经验法则告诉我们，应该将 40%(仅为参考，实际大小请参考自己操作系统中的内存资源和运行在操作系统中的应用情况)的可用内存分配给缓冲池(假设只有一种页面大小)，同时监控排序情况，并对 SHEAPTHRES_SHR 和 SORTHEAP 进行相应调整。

使用基于块(block-based)的缓冲池

连续读写频繁的 OLAP 查询可以得益于基于块的缓冲池。默认情况下，所有缓冲池都是基于页的，这意味着预取操作将把磁盘上相邻的页放入不相邻的内存中。而如果采用基于块的缓冲池，DB2 将使用块 I/O 一次将多个页读入缓冲池中，这样可以显著提高顺序预取的性能。

基于块的缓冲池由数据页和扩展数据块同时组成。CREATE 和 ALTER BUFFERPOOLSQL 语句的 NUMBLOCKPAGES 参数用于定义块内存的大小，而 BLOCKSIZE 参数则指定每个块的大小，即在一次块 I/O 中从磁盘读取的页的数量。

共享相同扩展数据块大小的表空间应该成为特定的基于块的缓冲池的专门用户。将 BLOCKSIZE 设置为等于正在使用缓冲池的表空间的 EXTENTSIZE 的整数倍。下面举一个创建基于块的缓冲池的例子：

```
    test2:/home/db2inst4$db2 create bufferpool block bp  size 40960 numblockpages
20480 blocksize 128
    DB20000I  The SQL command completed successfully.
```

确定分配多少内存给缓冲池内的块区要更为复杂一些。如果碰到大量的顺序预取操作，那么很可能会想要更多基于块的缓冲池。NUMBLOCKPAGES 应该是 BLOCKSIZE 的倍数，并且不能大于缓冲池页面数量的 98%。建议开始时先将其设小一点(不大于缓冲池总共大小的 15%或刚好 15%)，在后面还可以根据快照监视(snapshot monitor)对其进行调整。

3.4　DB2 V10 新特性——多温度存储器

多温度存储器是 DB2 V10 中新增加的特性之一，可以将不同访问频率的数据放置到不同的存储空间中。相对于以前的自动存储表空间和 DMS 表空间，这一特性增加了 STORAGE GROUP 的概念，更加方便 DBA 对不同热度的数据进行管理，降低了维护成本和硬件成本。

在数据大爆炸的今天，每天都会产生数以万计的数据，但是只有一小部分数据是需要频繁被访问的，大部分历史数据在数据库里是很难被访问检索到的，我们称频繁被访问的数据为热数据，不经常被访问的数据为冷数据。我们通常希望将热数据放到快速的存储上，而将冷数据放到更加廉价的存储上，但是随着时间的推移，冷热数据有可能经常发生变化，如何将冷热数据在不同类型的存储上进行迅速调整，对于 DBA 来说是巨大挑战。在 DB2 V9.7 中，通常做法是选择合适的存储来创建新的表空间，使用 ADMIN_MOVE_TABLE 存储过程将原表空间所有的表都移动到新的表空间中，期间涉及大量的参数配置、存储过程的调用，难以监控和诊断，是一项非常艰巨的任务。但是在 DB2 V10 里面，只需要通过 ALTER TABLESPACE 改变 STORAGE GROUP 即可。那么，STORAGE GROUP 到底是什么东西呢？

3.4.1　存储器组

存储器组(STORAGE GROUP)是可存储数据的存储器路径的指定集合。存储器组配置为表示可供数据库系统使用的不同存储器类。可对存储器组指定最适合于数据的表空间。只有自动存储表空间才使用存储器组。

一个表空间只能与一个存储器组相关联，但一个存储器组可与多个表空间关联。要管理存储器组对象，可使用 CREATE STOGROUP、ALTER STOGROUP、RENAME STOGROUP、DROP 和 COMMENT 语句。

1. 查看存储器组

可以采用 db2pd 工具来查看 STORAGE GROUP 的详细信息，后面加-storagegroup 参数：

```
    PNBDBB:/home/db2test$db2pd -d mydb -storagegroup

    Database Member 0 -- Database MYDB -- Active -- Up 1 days 08:00:53 -- Date
08/22/2012 17:15:47

    Storage Group Configuration:
    Address          SGID  Default  DataTag   Name
```

```
    0x07000000A0BC0000 0    Yes     0         IBMSTOGROUP
    0x07000001CD91FEA0 1    No      0         SG_HOT

    Storage Group Statistics:
    Address          SGID  State     Numpaths  NumDropPen
    0x07000000A0BC0000 0    0x00000000 1        0
    0x07000001CD91FEA0 1    0x00000000 1        0

    Storage Group Paths:
    Address          SGID  PathID   PathState    PathName
    0x07000000A0BC0120 0    0        InUse        /db2/mydb
    0x07000001CD91BEC0 1    1024     NotInUse     /db2/mydb/data_hot
```

也可以通过 SYSCAT.STOGROUPS 的系统视图来查看：

```
    db2 => select varchar(sgname, 12) as name, sgid, DEFAULTSG, cast(DEVICEREADRATE
as dec(5,1)) READRATE, cast(OVERHEAD as dec(5,3)) OVERHEAD, DATATAG from
SYSCAT.STOGROUPS

    NAME         SGID        DEFAULTSG READRATE OVERHEAD DATATAG
    ------------ ----------- --------- -------- -------- -------
    IBMSTOGROUP            0 Y           100.0    6.725       0
    SG_HOT                 1 N           100.0    6.725       0

     2 record(s) selected.
```

还可以通过系统函数 ADMIN_GET_STORAGE_PATHS 来查看 STORAGE GROUP 中包含的存储信息：

```
    db2 => select varchar(STORAGE GROUP NAME, 12) as NAME, STORAGE GROUP ID as SGID,
varchar(DB STORAGE PATH, 40) as STORAGE PATH, DB STORAGE PATH ID as PATH ID from
table (ADMIN GET STORAGE PATHS(NULL, -1)) where STORAGE GROUP NAME = 'SG HOT'

    NAME         SGID        STORAGE PATH                              PATH ID
    ------------ ----------- ----------------------------------------
--------------------
    SG HOT                 1 /db2/mydb/data hot                                  1024

     1 record(s) selected.
```

2. 创建存储器组

为了创建存储器组，可以使用 CREATE STOGROUP 命令来完成，基本语法如下：

```
    >>-CREATE--STOGROUP--storagegroup-name-------------------------->

     .-,--------------.
     V                |
>--ON---'storage-path'-+--●----------------------------------->

>--+----------------------------------+--●---------------------->
   '-OVERHEAD--number-of-milliseconds-'

>--+-----------------------------------------------+--●--------->
   '-DEVICE READ RATE--number-megabytes-per-second-'

>--+------------------------------------+--●-------------------->
   '-DATA TAG--+-integer-constant-+-'
               '-NONE-------------'

>--+----------------+--●------------------------------------><
   '-SET AS DEFAULT-'
```

在下面这个例子中，我们创建了名为 sg_cold 的存储器组，将/db2/mydb/data_cold 文件系统放到这个存储器组下：

```
PNBDBB:/home/db2test$db2 "create stogroup sg cold on '/db2/mydb/data cold'"
DB20000I  The SQL command completed successfully.
```

3. 修改存储器组

为了修改存储器组，可以使用 ALTER STOGROUP 命令来完成，基本语法如下：

```
>>-ALTER--STOGROUP--storagegroup-name-------------------------->


     .----------------------------------------------------------.
     |      .-,--------------.                                  |
     V      V                |                            (1)  |
>----+-ADD---'storage-path'-+------------------------+------+---><
     |      .-,--------------.                      |
     |      V                |                      |
     +-DROP---'storage-path'-+----------------------+
     +-OVERHEAD--number-of-milliseconds-------------+
     +-DEVICE READ RATE--number-megabytes-per-second-+
     +-DATA TAG--+-integer-constant-+----------------+
```

```
        |          '-NONE-------------'              |
        '-SET AS DEFAULT--------------------------------'
```

比如，要增加某个容器的路径到 sg_cold 存储器组里，可以采用下面的命令：

```
PNBDBB:/home/db2test$db2 "alter stogroup sg_cold add '/db2/mydb/data_cold2'"
DB20000I  The SQL command completed successfully.
```

查看新增加的存储路径：

```
PNBDBB:/home/db2test$db2pd -d mydb -storagegroup 2
Database Member 0 -- Database MYDB -- Active -- Up 1 days 08:18:01 -- Date 08/22/2012
17:32:55
Storage Group 2 Configuration :
Address            SGID  Default  DataTag    Name
0x07000001CD903820 2     No       0          SG_COLD
Storage Group 2 Statistics:
Address            SGID  State      Numpaths  NumDropPen
0x07000001CD903820 2     0x00000000 2         0
Storage Group 2 Paths :
Address            SGID  PathID   PathState    PathName
0x07000001CD8CC460 2     2048     NotInUse     /db2/mydb/data_cold
0x07000001CD8CB440 2     2049     NotInUse     /db2/mydb/data_cold2
```

4．删除存储器组

已存在的存储器组可以使用 DROP STOGROUP 命令来删除，在删除存储器组之前，必须确定是否有任何表空间使用该存储器组。如果有这样的表空间，那么在删除原始存储器组之前，必须更换这些表空间使用的存储器组并完成重新平衡操作。不能删除当前默认的存储器组。

```
PNBDBB:/home/db2test$db2 "drop stogroup sg_cold"
DB20000I  The SQL command completed successfully.
```

5．默认存储器组

创建数据库时，会自动创建名为 IBMSTOGROUP 的默认存储器组。但是，使用 AUTOMATIC STORAGE NO 子句创建的数据库没有默认存储器组。使用 CREATE STOGROUP 语句创建

的第一个存储器组变为指定的默认存储器组。只有一个存储器组被指定为默认存储器组。如果数据库有存储器组，那么在未显式指定存储器组的情况下创建自动存储管理的表空间时会使用默认存储器组。

可以使用 CREATE STOGROUP 或 ALTER STOGROUP 语句来指定默认存储器组。如果指定另一存储器组作为默认存储器组，那么对使用旧的默认存储器组的现有表空间没有影响。要改变与表空间相关联的存储器组，请使用 ALTER TABLESPACE 语句。

可以使用 SYSCAT.STOGROUPS 目录视图来确定哪个存储器组是默认存储器组：

```
   db2 => select varchar(sgname, 12) as name, sgid, DEFAULTSG, cast(DEVICEREADRATE
as dec(5,1)) READRATE, cast(OVERHEAD as dec(5,3)) OVERHEAD, DATATAG from
SYSCAT.STOGROUPS where DEFAULTSG='Y'
   NAME        SGID       DEFAULTSG READRATE OVERHEAD DATATAG
   ------------ ------------ ---------- --------- --------- --------
   IBMSTOGROUP           0 Y          100.0    6.725      0
     1 record(s) selected.
```

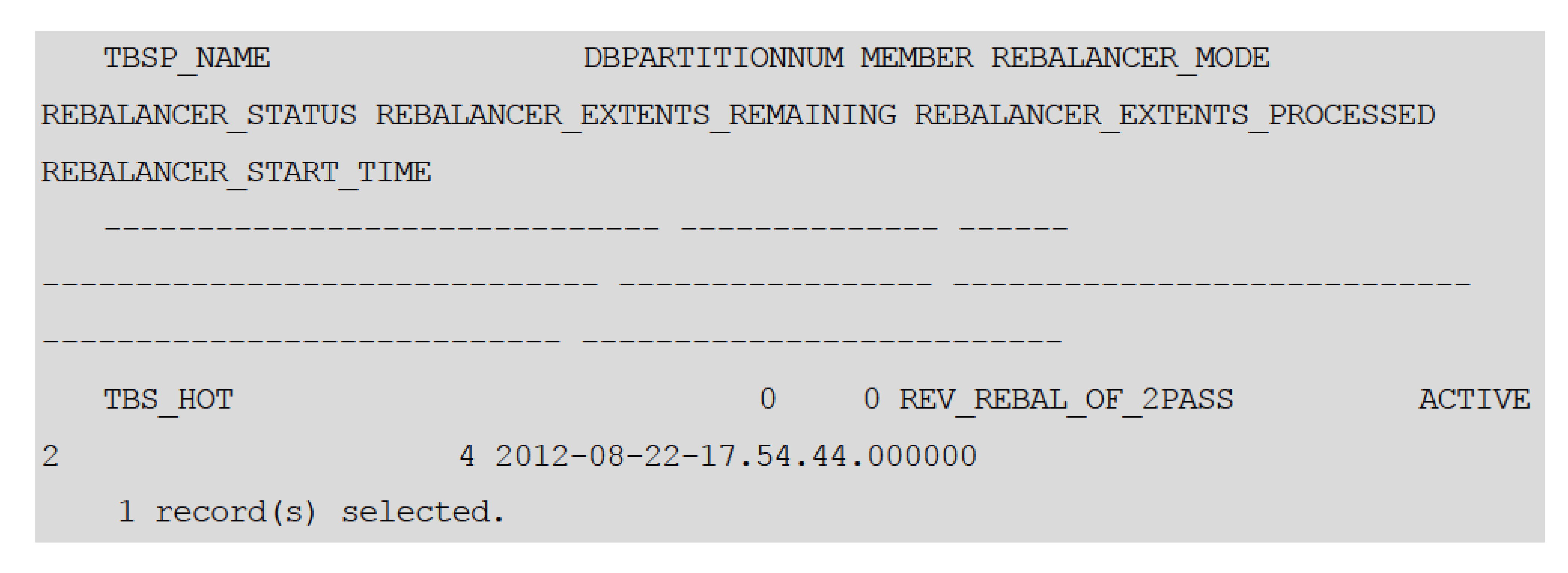
3.4.2　表空间与存储器组

创建表空间时，可以通过 CREATE TABLESPACE 语句指定表空间使用的存储器组。如果创建表空间时未指定存储器组，那么会使用默认存储器组：

```
   PNBDBB:/home/db2test$db2 "create tablespace tbs_hot using stogroup sg_hot"
   DB20000I  The SQL command completed successfully.
```

对于已经存在的表空间，可以使用 ALTER TABLESPACE 语句修改存储器组，但是如果更改表空间使用的存储器组，那么落实 ALTER TABLESPACE 语句时会发出隐式 REBALANCE 操作。这会将数据从源存储器组移至目标存储器组。可使用表监视函数 MON_GET_REBALANCE_STATUS 监视 REBALANCE 操作的进度：

```
   PNBDBB:/home/db2test$db2 "alter tablespace tbs_hot using stogroup sg_cold"
   DB20000I  The SQL command completed successfully.
   db2 => select varchar(tbsp_name, 30) as tbsp_name, dbpartitionnum, member,
rebalancer_mode, rebalancer_status, rebalancer_extents_remaining,
rebalancer_extents_processed, rebalancer_start_time from
table(mon_get_rebalance_status(NULL,-2)) as t
```



```
   TBSP_NAME                        DBPARTITIONNUM MEMBER REBALANCER_MODE
REBALANCER_STATUS REBALANCER_EXTENTS_REMAINING REBALANCER_EXTENTS_PROCESSED
REBALANCER_START_TIME
   ------------------------------ -------------- ------
------------------------------ ----------------- ----------------------------
---------------------------- --------------------------
   TBS_HOT                                  0      0 REV_REBAL_OF_2PASS          ACTIVE
2                      4 2012-08-22-17.54.44.000000
    1 record(s) selected.
```

使用同一存储器组的所有表空间可以有不同的 PAGESIZE 和 EXTENTSIZE 值。这些属性与表空间定义相关，与存储器组无关。

3.5　本章小结

本章讲解了如何设计、规划和创建数据库、表空间和缓冲池，并且介绍了 DB2 V10 新增加的“多温度存储器”特性。良好的设计是整个应用系统高性能运行的基石。希望你能了解数据库的特性，用最适合业务需求的技术来进行数据库的物理设计和逻辑设计。随着数据库技术的发展，数据库中有很多新技术克服了以往技术的一些缺点，所以你必须了解这些新技术并能够合理地运用。就像在 DB2 V9 中建议大家创建基于自动存储的表空间，并且在创建表空间时尽量创建大型表空间。



访问数据库

当数据库创建后，我们应当如何访问数据库呢？可以通过 DB2 数据库本身提供的管理工具——GUI 图形化界面来访问数据库；也可以通过类似 DOS 的命令行窗口 DB2 CLP 来访问数据库；另外，还可以通过编写程序来访问数据库。需要注意的是，如果想在远程客户端访问 DB2 数据库，就必须配置 DB2 服务器通信和客户端通信。

本章将对如何访问数据库进行探讨，主要讲解如下内容：

- DB2 GUI 图形化界面
- DB2 CLP 命令行窗口
- 配置 DB2 服务器通信
- 配置客户端通信

4.1 访问 DB2

图 4-1 展示了访问 DB2 数据库的各种接口。可以通过 DB2 CLP 访问数据库；也可以通过 DB2 的管理工具——图形化界面访问数据库；还可以通过应用编程接口编写程序的方式访问数据库。如果远程客户端需要访问 DB2 数据库，就需要配置服务器通信和客户端通信。下面将分别讲解这些接口。

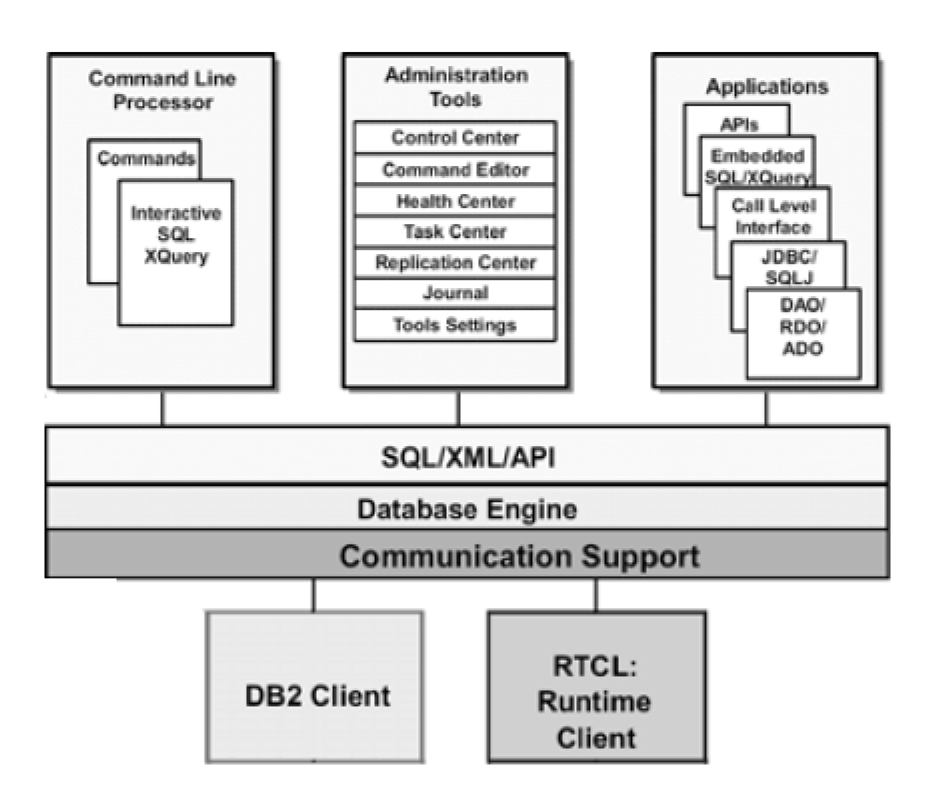


图 4-1　访问 DB2 数据库的接口

4.2　DB2 图形化操作环境

在进行数据库设计时，常常需要在 DB2 图形化环境下进行操作。DB2 图形化环境提供了大量的图形化工具，因此，在创建 DB2 数据库之前，就非常有必要了解并熟悉这一图形化操作环境。虽然从 V9.7 开始，DB2 不推荐使用控制中心工具和 DAS，转而推荐使用 IBM Data Studio 等工具，但是由于控制中心工具和 DAS 等的应用目前也比较广泛，大部分用户还没有转向 Data Studio 等工具的使用，所以本章还是以传统的 DB2 图形化环境为例，介绍 DB2 的相关概念和知识。

图 4-2 对 DB2 的图形化工具进行了总结。

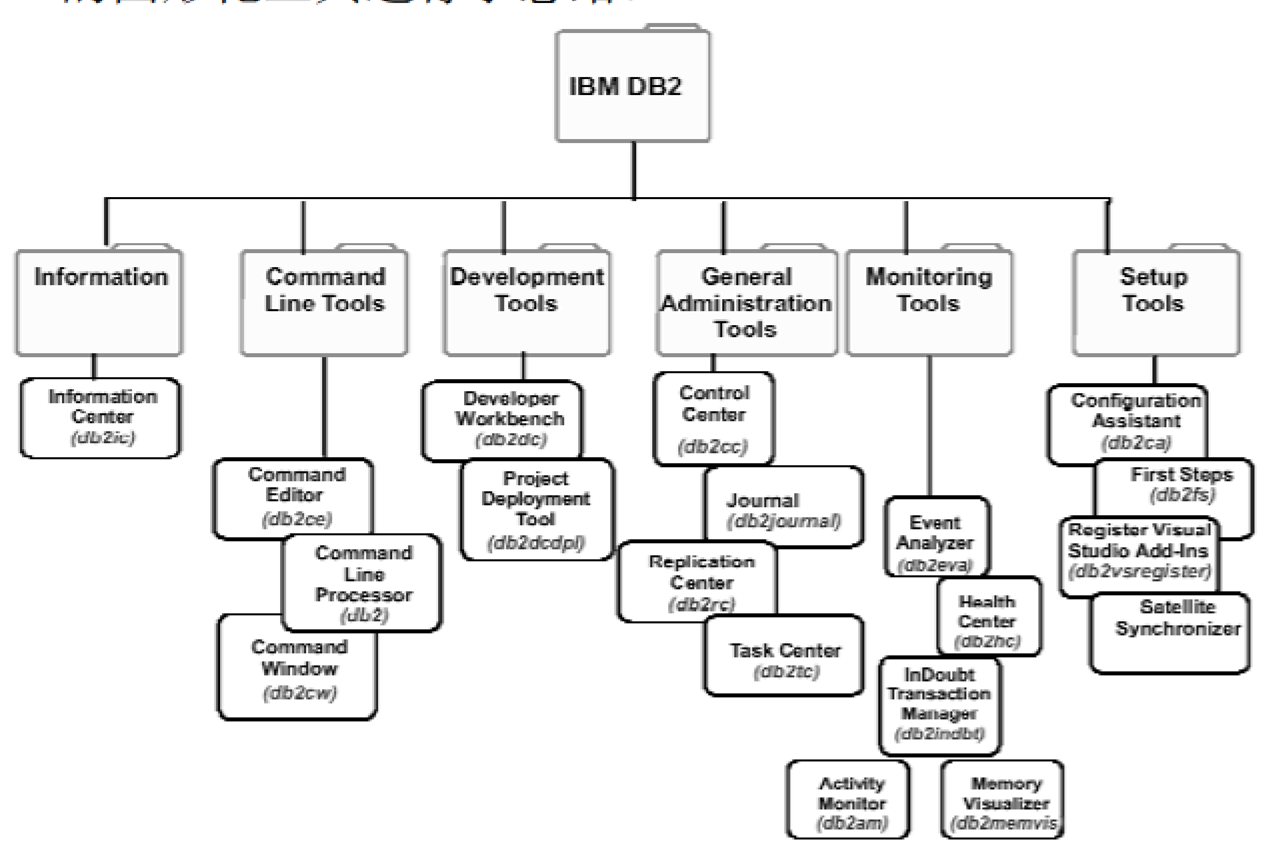


图 4-2　DB2 的图形化工具

图 4-3 是 Windows 中调用 DB2 图形化工具的菜单，下面将分别详细讲解每一种图形管理工具。

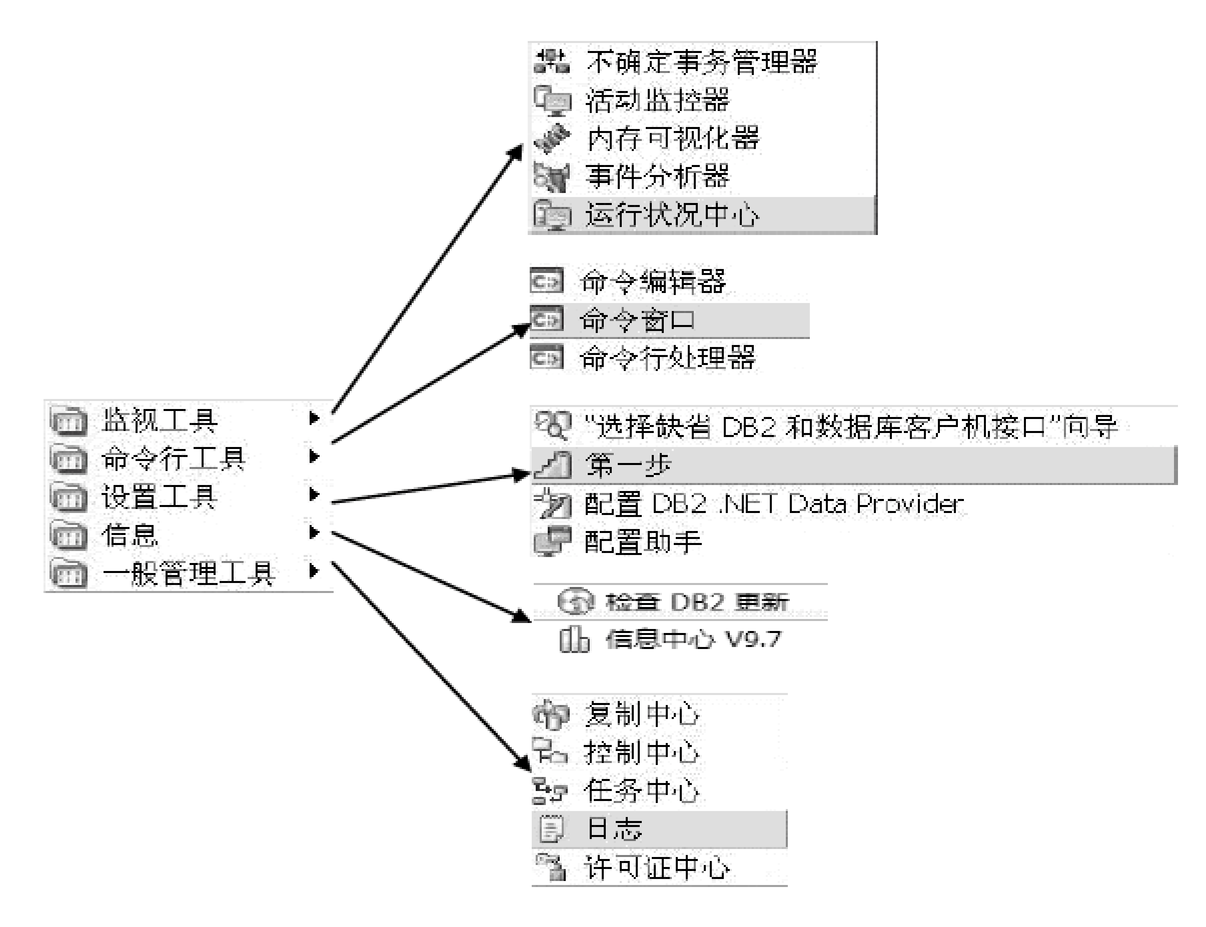


图 4-3　调用 DB2 图形化工具的菜单

控制中心

控制中心(Control Center)是 DB2 服务器的中心管理点，是 DB2 管理工具的核心，绝大多数管理任务和对其他管理工具的存取都可以通过控制中心来完成，如图 4-4 所示。



图 4-4　控制中心

在 Windows 平台上，可以依次从“开始”菜单选择“IBM DB2”，接着选择“一般管理工具”，最后选择“控制中心”来启动控制中心。也可以使用 db2cc 命令来启动控制中心。

控制中心依赖于数据库管理服务器(Database Administration Server，DAS)。DAS 帮助控制中心调度作业在数据库服务器上的运行，并管理远程数据库服务器上的对象以及其他任务。

控制中心主要为数据库管理员提供下列功能：

- 添加 DB2 系统、数据库和数据库对象到对象树中。
- 管理数据库对象，包括创建、更改和删除数据库、表空间、表、视图、索引、触发器和模式，还可以管理系统、实例、用户、组、别名、用户定义类型(UDT)、用户定义函数(UDF)、应用程序、程序包以及复制对象。
- 管理数据。可以加载、导入或导出数据、重组数据并收集统计信息。
- 调度作业来自动运行。
- 备份和恢复数据库。
- 配置实例和数据库。
- 分析查询和可视化解释(Visual Explain)。使用可视化解释分析查询，查看访问计划。
- 监视并调优性能。可以打开统计表(查看查询的执行路径)，启动事件和快照监视、生成数据库对象或命令的 SQL 或 DDL 并查看 DB2 对象之间的关系。
- 启动其他工具，比如命令编辑器和健康中心。
- 更改整个控制中心用于显示菜单和文本的字体。

第一步

在完成 DB2 服务器端的软件安装之后，在默认配置下，系统会自动启动“第一步”，如图 4-5 所示。此时，DB2 服务器上并不存在任何数据库，但用户可以通过单击“第一步”窗口中的“创建 SAMPLE 数据库”链接来创建自己的第一个 DB2 数据库。

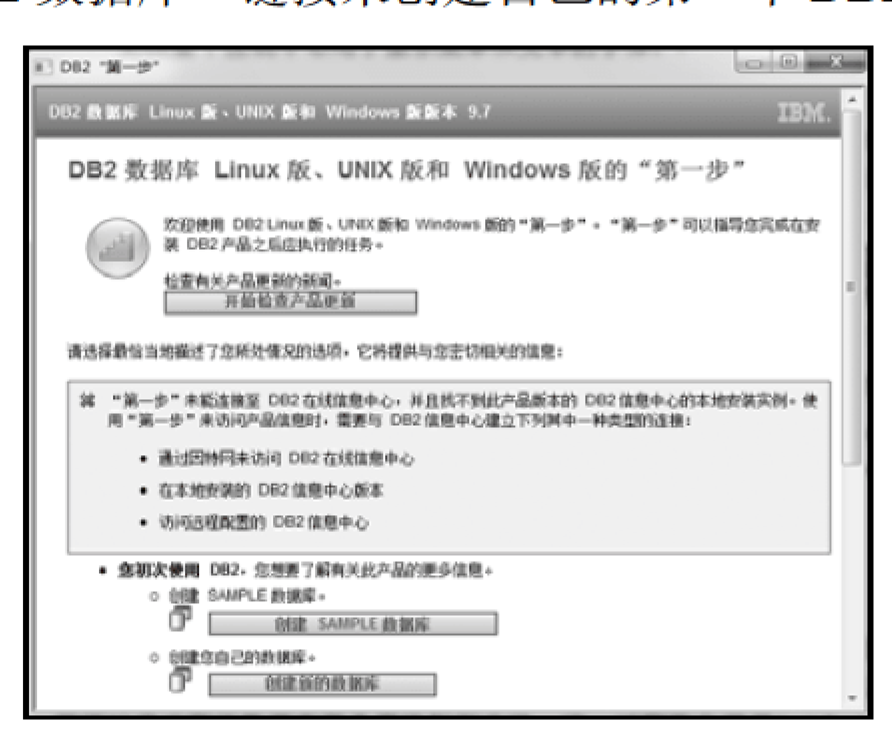


图 4-5 “第一步”窗口

配置助手

客户端如何能够识别远程数据库服务器呢？DB2 采取的办法是将远程数据库服务器端的节点信息、数据库信息分别写入客户端本地的 SQLNODIR 文件和 SQLDBDIR 文件，这样就可以将远程的数据库服务器映射到本地，这一过程称为编目(catalog)。而在编目之前，客户端必须能够与 DB2 服务器通信，客户机系统的数据库管理员必须建立数据库管理系统与 DB2 服务器的通信，建立的方式则取决于操作系统和 DB2 服务器采用的通信协议。

DB2 配置助手(CA)可以用来对客户端应用程序使用的数据库服务器进行配置和维护，可以在客户端通过图形化的方式对远程数据库服务器端的节点和数据库进行编目，使用户免受命令行方式下对数据库进行手工编目之苦。如图 4-6 所示，配置助手(CA)可以用来维护客户端当前连接的 DB2 数据库配置参数，维护当前连接的 DB2 数据库管理服务器参数，配置新的数据库连接，绑定应用程序以及导入和导出配置信息。

在 Windows 平台上，可以依次从“开始”菜单选择“IBM DB2”，接着选择“设置工具”，最后选择“配置助手”来启动配置助手。也可以使用 db2ca 命令来启动配置助手。

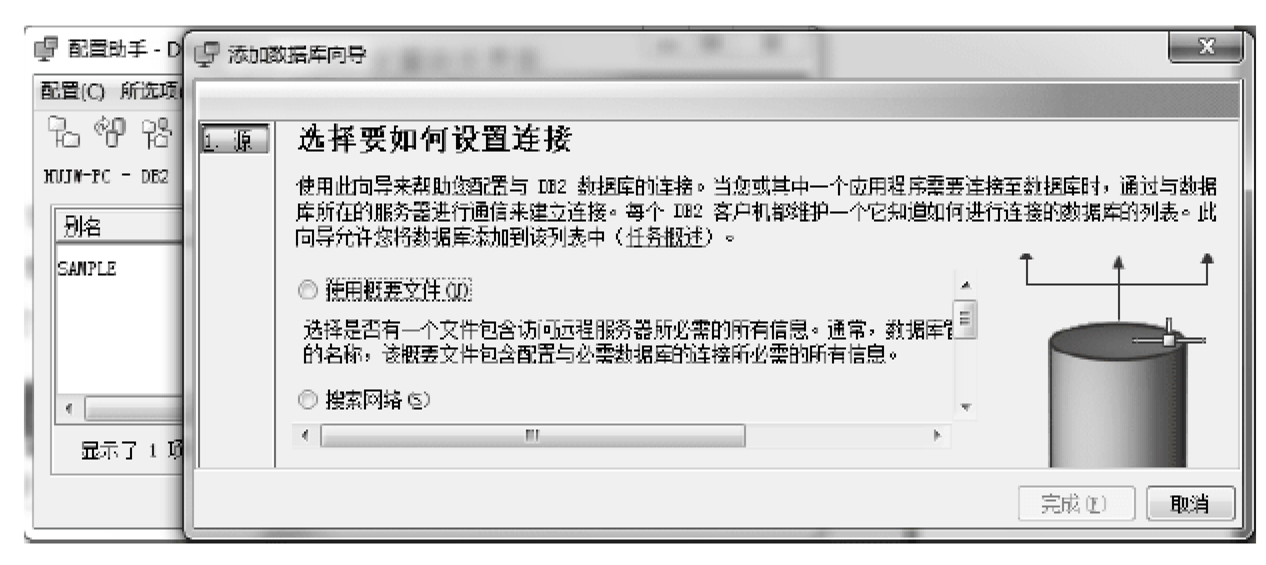


图 4-6　DB2 配置助手的界面

复制中心

复制中心用于管理 DB2 数据服务器和其他关系数据库(DB2 或非 DB2)之间的复制。可以使用该工具创建复制定义和管理 Capture、Apply 和 Monitor 程序。可以在复制中心设置 SQL 复制、Q 复制等，如图 4-7 所示。

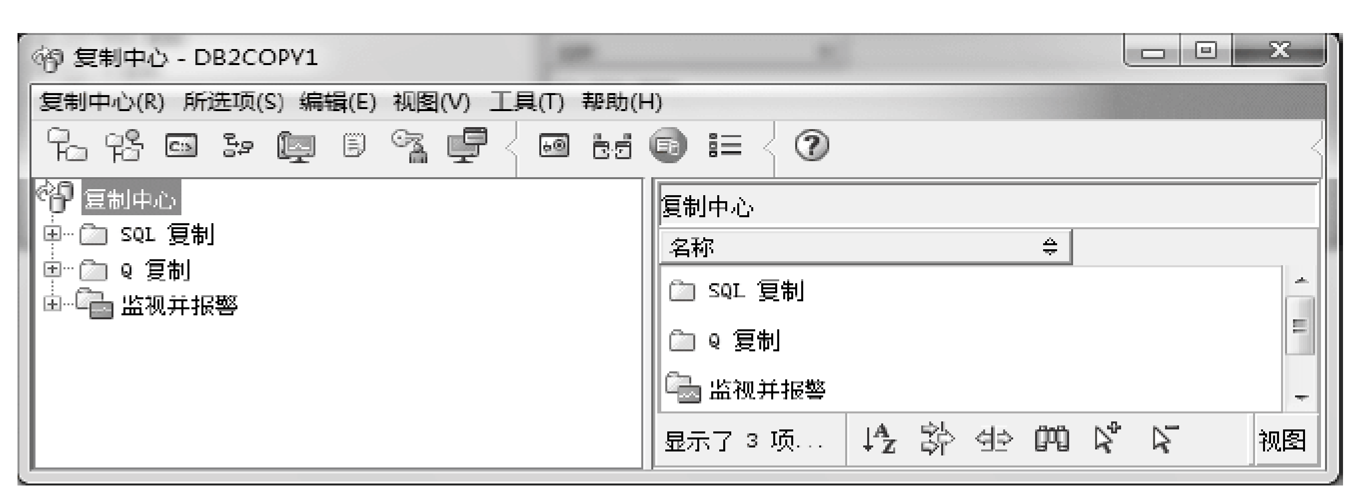


图 4-7　复制中心的功能

命令编辑器

命令编辑器是用来输入 DB2 命令的图形化工具，如图 4-8 所示。命令编辑器就是图形化的命令处理器(CLP)，可以在命令编辑器内输入 DB2 命令或调用现成的命令脚本，执行后可以查看输出结果，相当于 DB2 CLP 命令行的图形化界面实现。

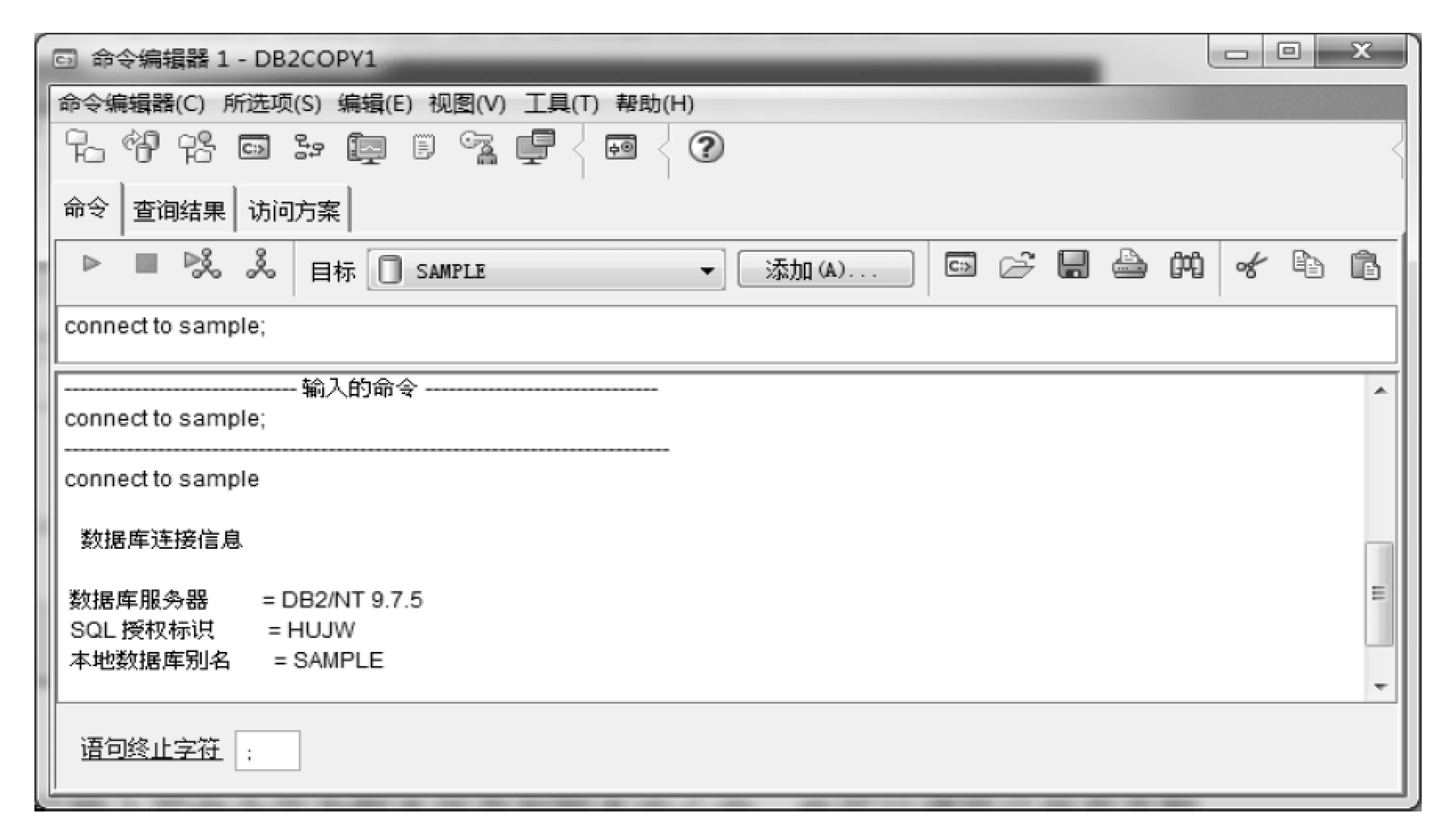


图 4-8 命令编辑器

命令编辑器可以将已输入的命令作为脚本保存到脚本中心，也可以调用已保存在脚本中心内的脚本。另外，命令编辑器特别有用的功能之一就是，用户可以用来查看 SQL 语句的存取计划。存取计划中包含了 SQL 语句执行情况的统计结果，这样用户就可以通过命令编辑器为 SQL 语句生成存取计划，并按照可视化的形式表现出来。

任务中心

任务中心可以用来安排、运行任务并通知人们已完成任务的状态。任务是一种附带相关的失败或成功条件、调度计划和通知的脚本，脚本中可以包含 DB2 命令、SQL 语句或操作系统命令。

任务中心中还可以创建组合任务以根据多个任务的结果来定义操作。组合任务与任务中心中的其他任务不同，因为没有任何命令脚本与组合任务直接关联。组合任务包含了已在任务中心中定义的任务。创建组合任务的优点是可创建依赖于多个任务结果的任务操作。任务中心如图 4-9 所示。

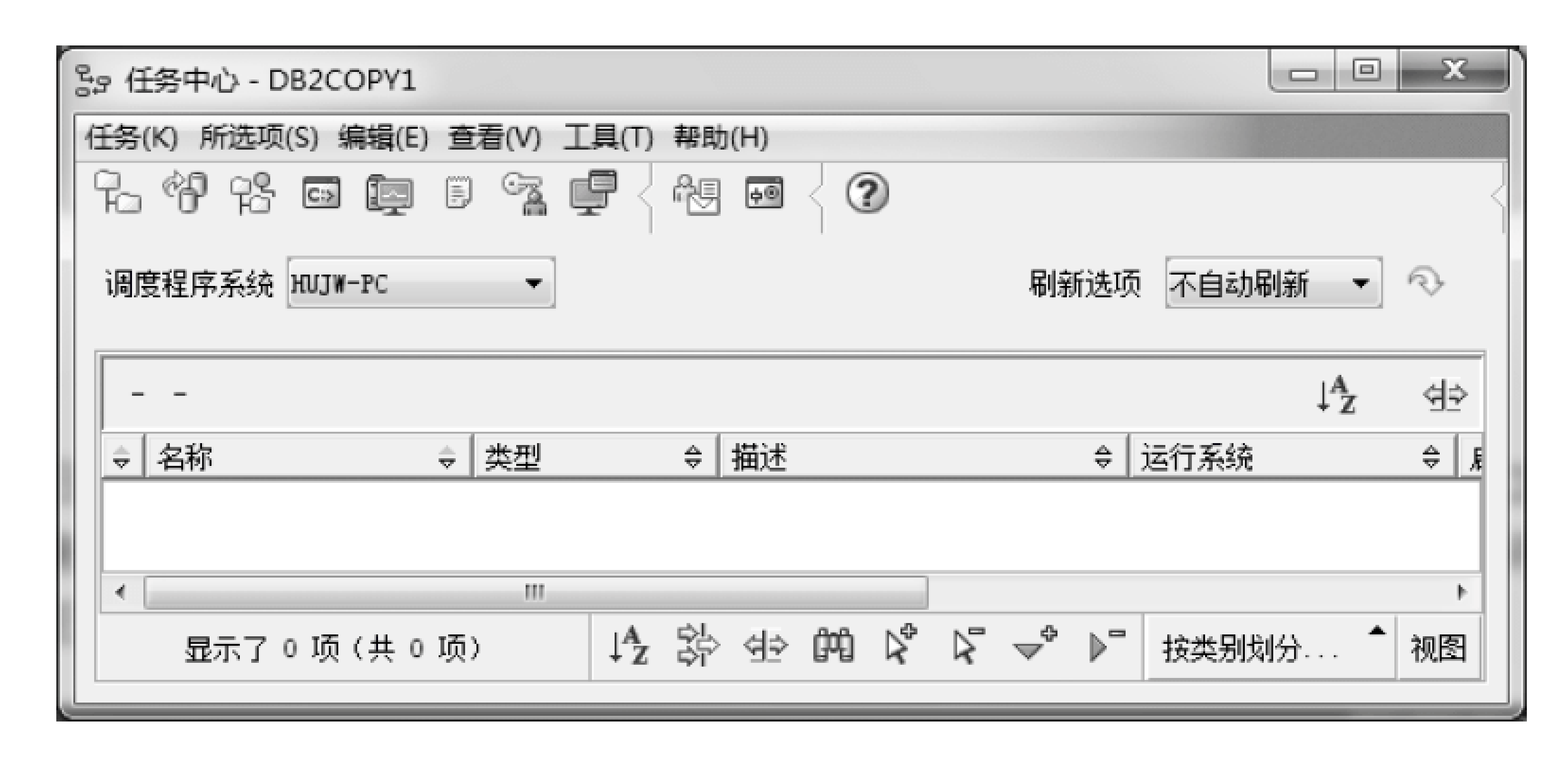


图 4-9　任务中心

日志

利用日志能够监视作业和查看结果，如图 4-10 所示。可以通过从控制中心的工具栏选择日志图标来启动日志。从日志中还可以显示恢复历史和 DB2 警告消息。日志允许监视暂挂的作业、正在运行的作业和作业历史，查看结果，还能显示 DB2 消息记录。



图 4-10　“日志”窗口

许可证中心

许可证中心显示 DB2 许可证的状态和安装在系统中的 DB2 产品的使用情况，如图 4-11 所示。另外，还可用来配置系统以便进行适当的许可证监视。可以用许可证中心来添加新的许可证，设置并发的用户策略，将先试后买的许可证升级为生产版许可证，用户也可以浏览当前安装在 DB2 系统中的许可证信息，例如产品名称、产品的版本、过期时间，以及允许的用户数目等信息。此外，还可以通过在命令行使用 db2licm 命令来维

护 DB2 许可证。

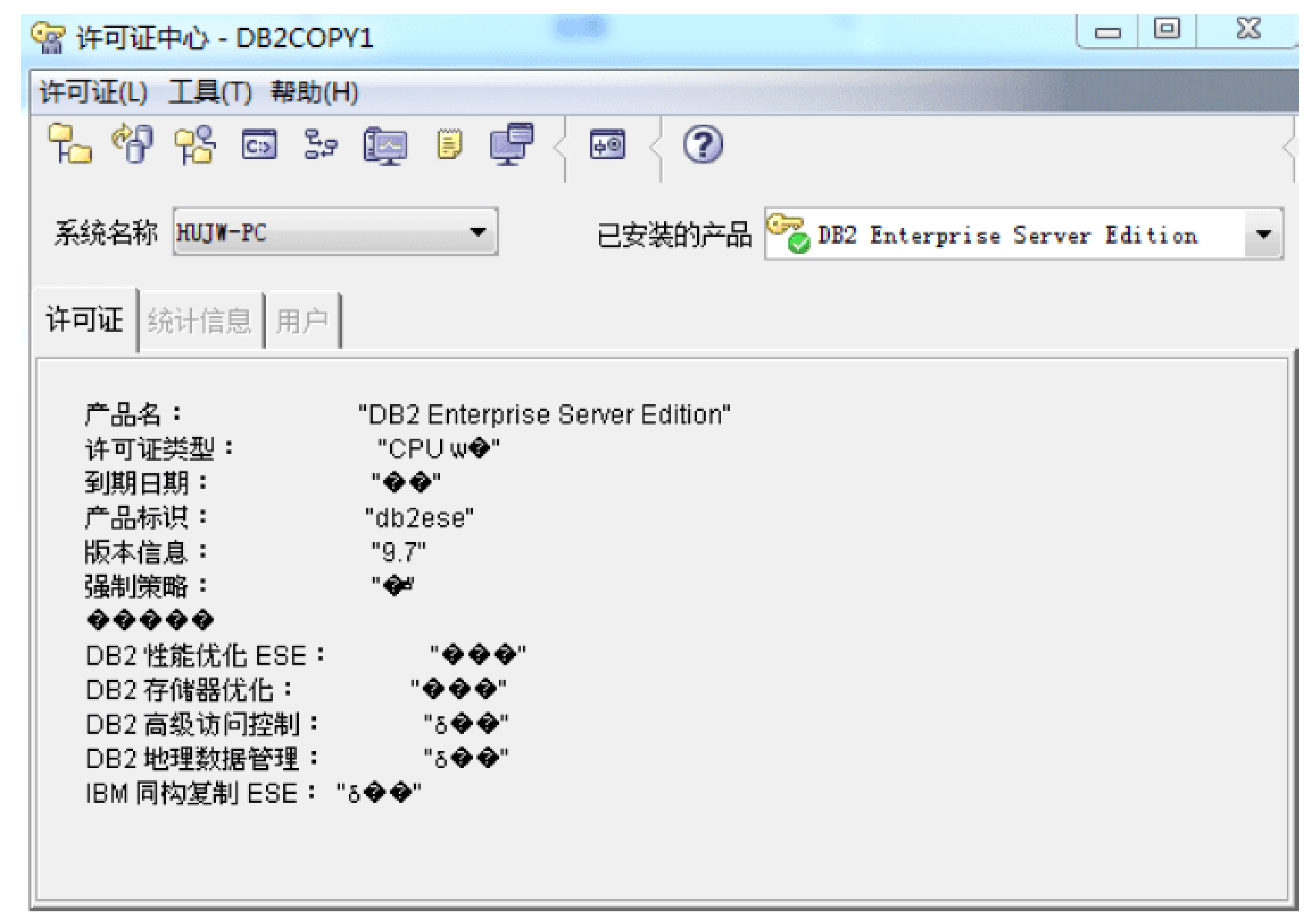


图 4-11 “许可证中心”窗口

由于操作系统语言的原因，个别中文未显示正确，可以通过在命令行使用 db2licm 命令来查看 DB2 的许可证信息，如图 4-12 所示。

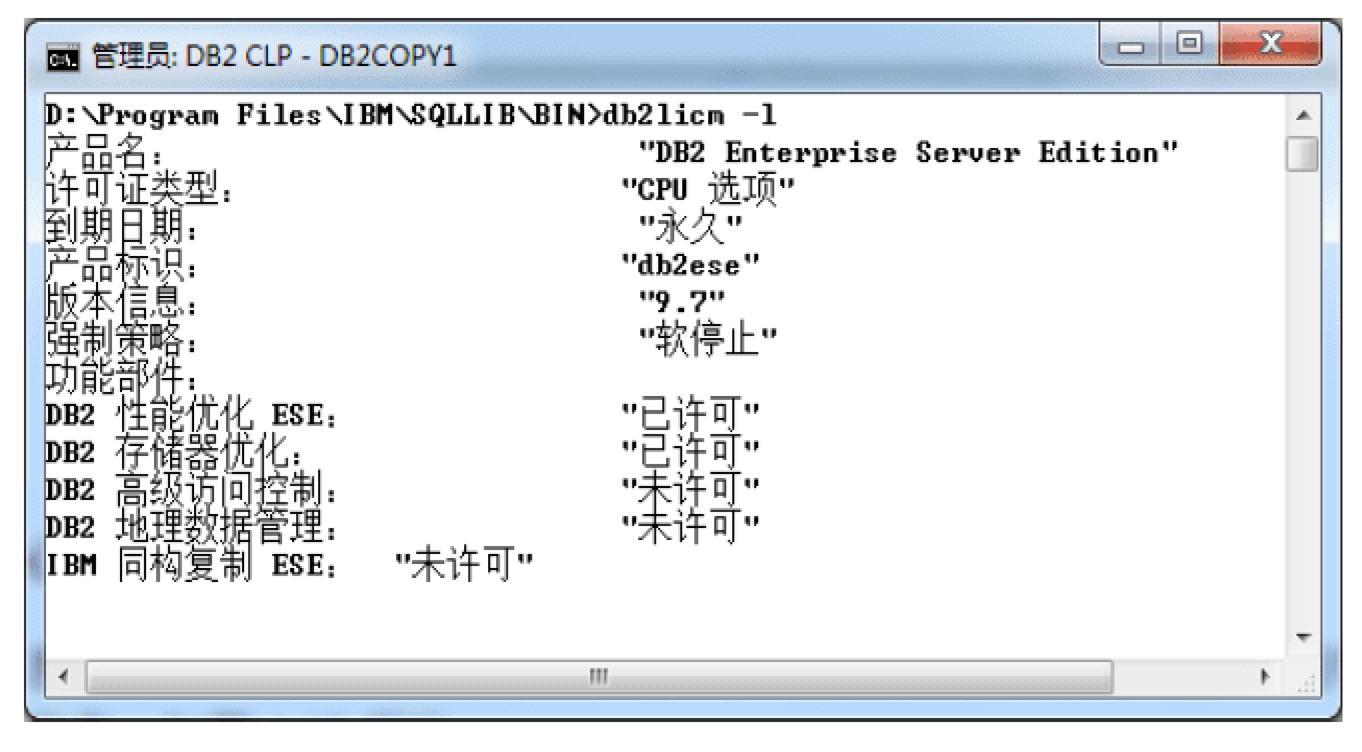


图 4-12 “命令行 db2licm”窗口

信息中心

使用信息中心可以查找关于任务、书籍、参考资料、故障排除、样本程序以及相关 Web 站点的信息，如图 4-13 所示。



图 4-13 信息中心

信息中心提供了丰富的 DB2 信息。在 Windows 环境中，可以从控制中心或从“开始”菜单启动信息中心。也可以使用 db2ic 命令快速启动信息中心。

内存可视化器

使用内存可视化器监视 DB2 数据库的内存使用情况，如图 4-14 所示。



图 4-14 内存可视化器

不确定事务管理器

使用不确定事务管理器处理不确定的全局事务，如图 4-15 所示。例如，中断的通信会让事务做好准备，但还不会提交或回滚。通常用于诊断分布式数据库的两阶段提交。

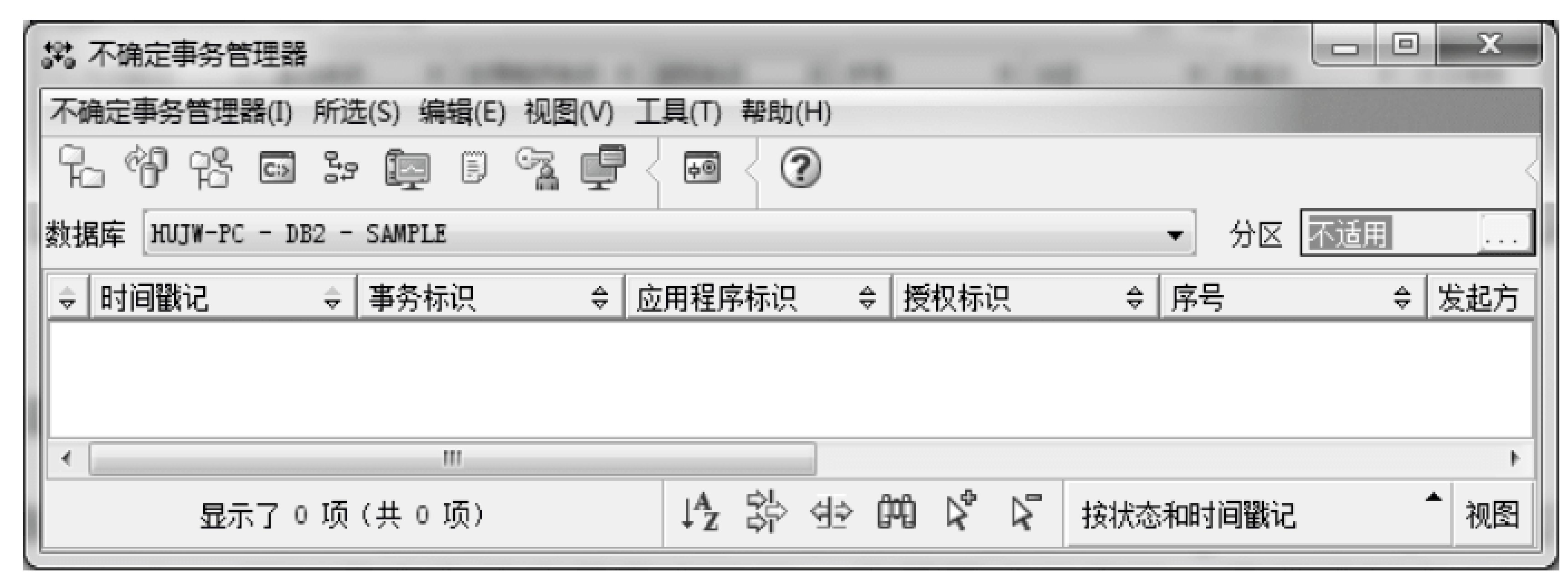


图 4-15　不确定事务管理器

活动监视器

DB2 UDB V8.2 中增加了一个新的图形化工具，称为 Activity Monitor(活动监视器)。Activity Monitor 可以用来监视应用程序性能、应用程序并发性、资源消耗和 SQL 语句的使用情况，如图 4-16 所示。活动监视器可以帮助用户诊断数据库性能问题(比如锁等待状态)，以及调优查询来优化对数据库资源的使用。Activity Monitor 还提供 DB2 自动生成的许多报告。

其实，活动监视器本质上是对底层调用的一些监控命令和函数的封装，只不过是以图形化的直观方式显示。在这里，建议大家熟练掌握监控数据库性能的手工命令，这是因为在实际的数据库生产环境中大多数没有配置图形化环境，并且真正的高手很少依赖图形化界面。尽管如此，活动监视器仍不失为好的监控维护工具。



图 4-16　Activity Monitor -运行时分析界面

运行状况中心

该工具可以显示客户机上编目的所有实例的数据库系统健康运行状况。“运行状况中

心”收集编目实例的信息，并在所有中心或“健康中心”主视图中提供潜在或现有问题的文本或图形通知。可以使用界面来查看每条警告的详细信息，如图 4-17 所示。该工具对如何解决问题提出建议，并提供了界面以应用解决方案。“运行状况中心”使用简易，DB2 的初学者也能够使用它成功判断问题并进而获取解决方案。



图 4-17 运行状况中心

事件分析器

这个管理工具只在 Windows 平台上才有，可以用来对事件监视器的监控结果进行图形化分析，如图 4-18 所示。

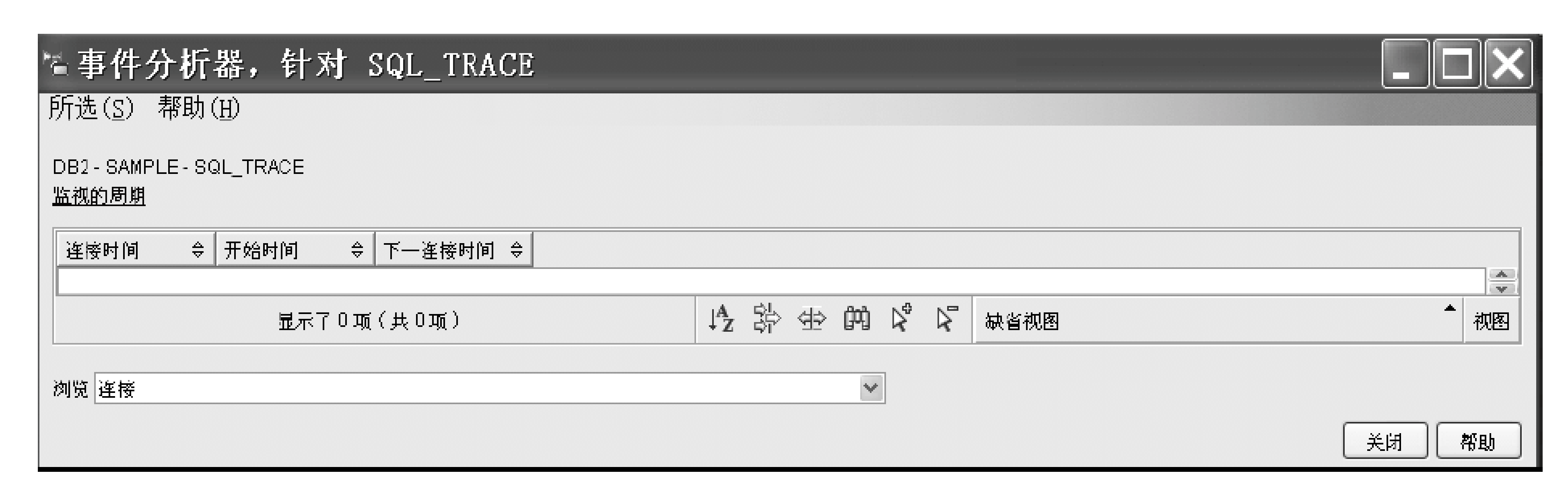


图 4-18 事件分析器

4.3 DB2 CLP 处理程序

4.3.1 DB2 CLP 简介

DB2 命令行处理器听起来没什么特别之处，但实际上却是 DB2 的接口，充分体现了

DB2 的威力，以及 DB2 的简单性和通用性。命令行使我们想起了 UNIX 中的 Telnet，还有经常使用的 DOS 命令。DB2 Command Line Processor(DB2 CLP)是所有 DB2 产品都必备的工具，可以使用这个应用程序运行 DB2 命令、操作系统命令或 SQL 语句。虽然最终用户可能永远不会使用 CLP 来访问他们的数据，但是对于 DBA 或者应用程序的编程人员来说，CLP 在工具箱中则是最基本的工具——就像是在所有场合都适用的燕尾服。数据库通常是核心应用最关键的组成部分，如果数据库层面发生故障，问题的判定、解决相对会比较困难。而 CLP 正是 DBA 访问数据库，进而诊断、排除故障的入口。

注意：

其实每个数据库都提供类似 DB2 CLP 的命令行接口工具，例如 Oracle 的 SQL*PLUS、Informix 的 DBACCESS、Sybase 的 isql。

4.3.2 DB2 CLP 设计

CLP 从架构上来说由两个进程组成：

- 前端进程(或是在 Windows 中的线程)，用于处理与操作系统命令提示符的通信。
- 后端进程，用于处理与数据库的通信。这确保在连接到 DB2 之后，如果用 Control-C 或 Control-Break 中止来自某个大型选择查询(例如 SELECT* FROM SYSCAT.TABLES)的输出，那么会顺利中止输出，而不会断开与 DB2 的连接。

命令窗口与 Windows 中 CLP 的比较

在 Linux 和 UNIX 环境下，可以通过命令行界面直接输入 db2 操作命令来调用 DB2 CLP；而在 Windows 环境下则不行，需要在命令窗口先执行 db2cmd 命令或 db2cw 命令以启动 CLP。为什么在 Windows 环境下，DB2 要求使用 DB2CMD.EXE 启动 CLP 呢？这是因为在 UNIX 和 Linux 中，前端进程和后端进程之间的连接非常简单：如果父进程死亡，那么所有子进程都将被操作系统终止。而在 Windows 中，父线程在死亡时并不会终止其子线程。因此，DB2 使用 cookie 来为 Windows 中的 CLP 连接前端线程和后端线程。这就要求 CLP 必须通过 DB2CMD.EXE 来启动。这样做可以确保如果杀死了父线程，那么子线程也不会保留下来，从而避免了资源的浪费。如果 DB2 不采用这种技术，就会产生大量的 phantom 线程。

DB2 中有两种不同的处理器：DB2 命令行处理器(DB2 CLP)和 DB2 命令窗口(DB2 CW)。有人喜欢用同样的名字 DB2CLP 称呼它们，因为它们在 Windows“开始”菜单中有相同的图标。对于除 Windows 之外的所有操作系统，DB2 CW 是在操作系统的本机 CLP 中内置的。在 Windows 环境中，可以在 Windows 命令提示中输入 db2cmd 或 db2cw 命令来启动 DB2CW，还可以在 DB2 的“命令行工具”菜单中选择“命令窗口”来启动 DB2CW；

而 DB2 命令行处理器则可以通过在 DB2 的“命令行工具”菜单中选择“命令行处理器”来启动，或者在 DB2 命令窗口中输入 db2 命令来启动，这时将进入到交互模式，这种模式称作 DB2 交互式 CLP，并且会创建如下所示的特殊提示符：

```
db2 =>
```

在 DB2 交互式 CLP 下，不必在执行 DB2 命令时加上 DB2 前缀。但这时必须在执行操作系统命令时加上惊叹号(!)前缀。例如，如果想运行 dir 命令，就必须输入!dir。

图 4-19 显示了通过 DB2 CW 以非交互方式输入的一个命令。

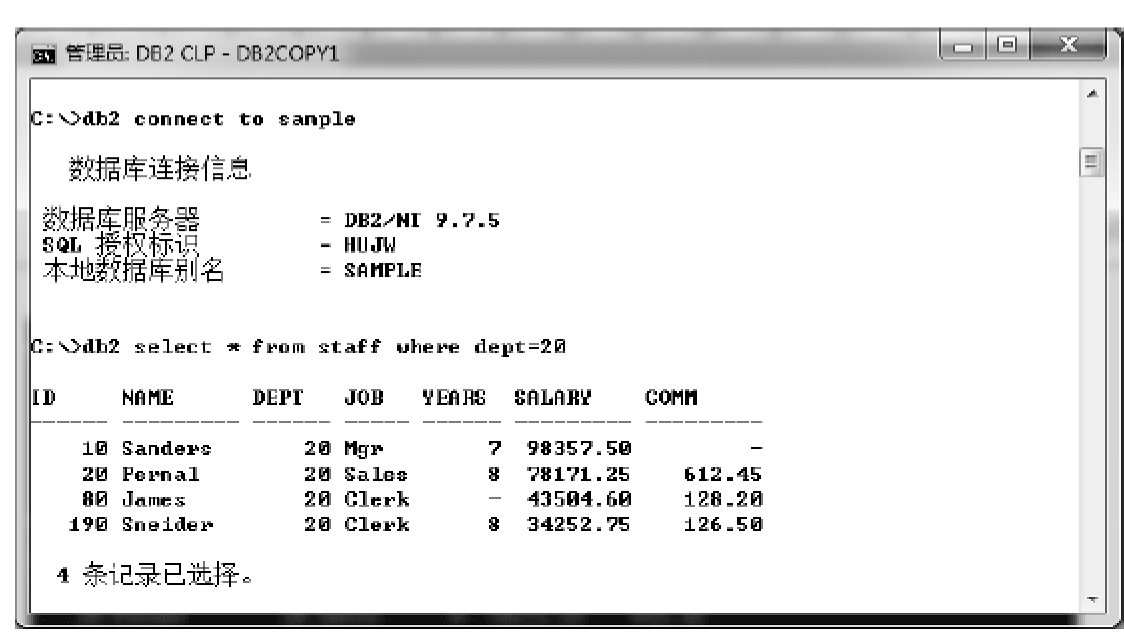


图 4-19　用 DB2 CW 输入命令

注意，运行这个 DB2 命令时，必须输入关键字 db2 前缀。如果不这么做，操作系统就会认为这是操作系统命令，将返回错误。如果使用 DB2 交互式 CLP，就不需要这么做，如图 4-20 所示。

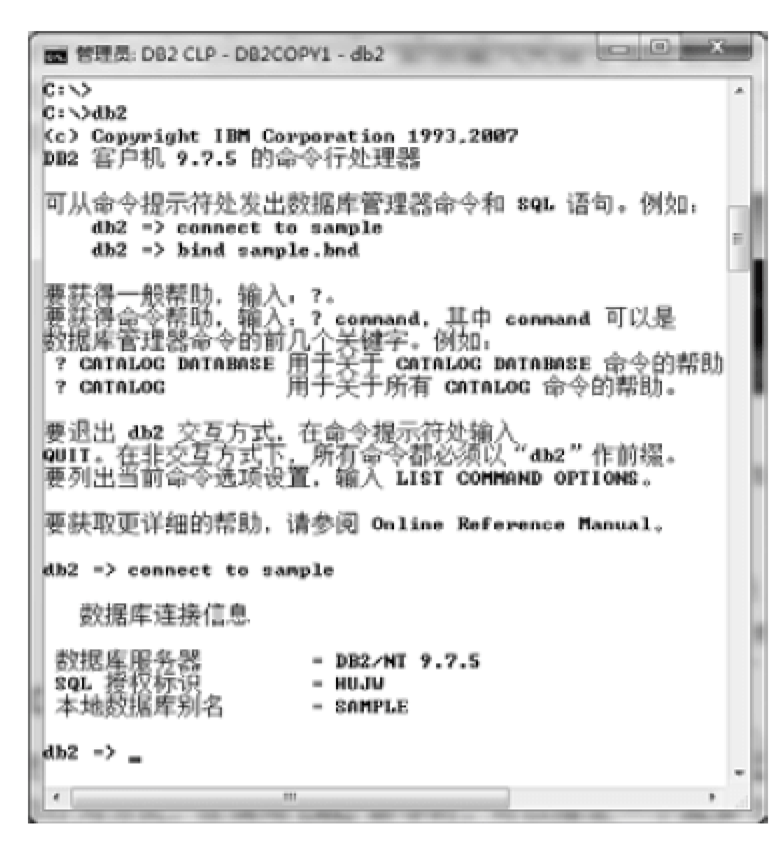


图 4-20　用 DB2 CW 以交互方式输入命令

4.3.3 DB2 CLP 命令选项

在使用 DB2 CLP 处理程序时，可以使用命令行选项修改处理过程或处理中输入的语句和命令的行为方式。在调用 DB2 命令时，可以指定一个或多个处理程序选项。常用的一些选项如下：

- 可以使用 c 标志定义每条语句的自动提交。
- 可以使用 v 标志定义在命令执行的同时在屏幕上输出。
- 可以使用 s 标志定义执行命令序列时碰到错误停止执行。
- 可以使用 z 标志定义把命令执行期间的结果输出到文件中。
- 可以使用 f 标记定义提供 DB2 命令和 SQL 语句的输入文件中。
- 可以使用 t 标记定义语句末尾的结束字符(默认字符是“;”)。如果不想用“;”结尾，而想用@结尾，那么可以使用-td@方式读取输入文件。
- 可以通过在 DB2 处理程序中输入 db2 list command options 命令或 db2 ? options 命令获得所有有效选项的列表。如图 4-21 所示，运行这个命令，会看到 19 个以上的选项。



图 4-21　各种 DB2 CLP 选项

有 3 种修改 DB2 处理程序选项的方法：

- 可以通过 db2set DB2OPTIONS 直接修改注册变量，这会永久改变 DB2 处理程序选项的值。例如：

```
C:\>db2set db2options=-c
```

- 可以使用 update command options 命令修改 DB2 处理程序选项，这会在会话级改变 DB2 处理程序选项的值。优先级高于使用 DB2set DB2OPTIONS 设置，并且将会覆盖在注册表级建立的任何设置。例如：

```
db2=>update command options using c on
DB20000I  UPDATE COMMAND OPTIONS 命令成功完成
db2=>
```

- 在输入 DB2 命令时指定命令行标志，这将会在语句级改变 DB2 处理程序选项的值。优先级高于使用 DB2set DB2OPTIONS 设置，也高于使用 update command options 命令设置，并且将会覆盖在注册表级和会话级建立的所有设置。例如：

```
db2 -c command or statement...
```

正如上面的图 4-21 中所示，在使用指定命令行标志打开选项时，应该在对应的选项字母前面加上减号(-)；例如，要打开自动提交特性(这是默认的)。

要关闭选项，可以在选项字母前后加上减号(-c-)，或者在前面加上加号(+)。再解释一下前两句话，因为这儿可能有点儿混乱：在标志前面放上减号会打开选项；在标志前面和后面都放上减号，或者在标志前面放上加号会关闭选项。这么说也许还是不太明确，因为这可能导致混淆，我们用自动提交选项举个例子。

一些命令行选项是默认打开的，其他的是默认关闭的。前面的解释(和后面的示例)描述默认打开的命令行选项的行为和效果。如果命令行选项是默认关闭的，那么使用相反的逻辑。在默认情况下，自动提交特性是打开的(-c)。这个选项指定每条语句是否自动提交或回滚。

如果一条语句成功了，就和前面执行的未提交的(关闭了自动提交所致)的所有成功语句一起提交。但是，如果失败了，就和前面执行的未提交的(关闭了自动提交所致)所有成功语句一起回滚。如果这条语句关闭了自动提交，就必须显式地执行提交或回滚命令。

在图 4-22 中，在命令行上修改自动提交特性的值以演示这个过程。

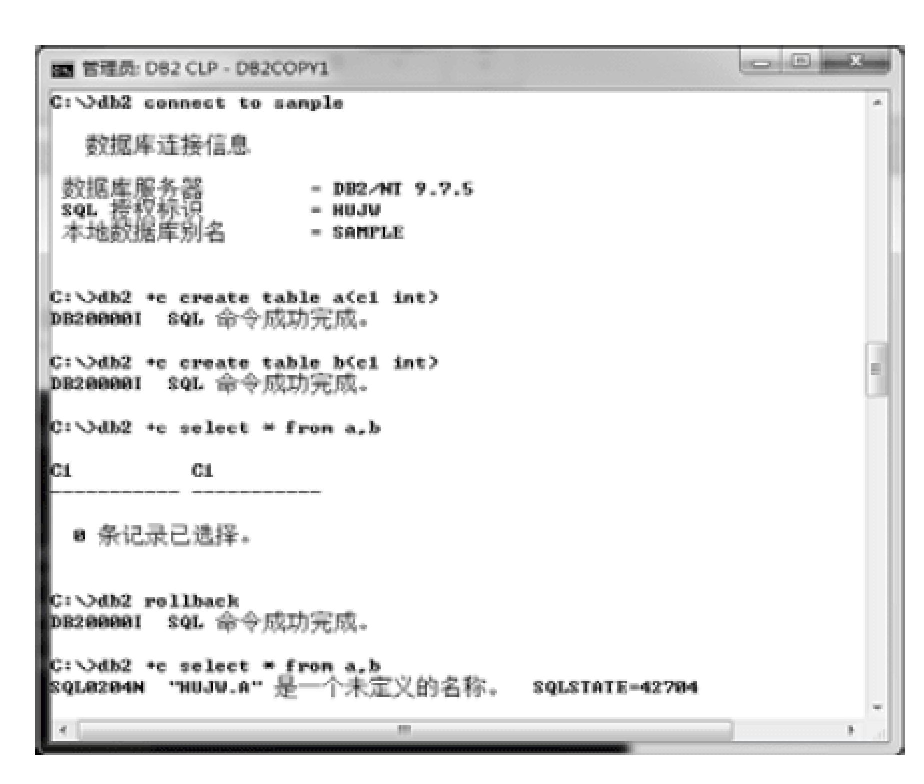


图 4-22　在运行时修改命令行选项

那么，发生了什么情况？首先，创建名为 A 的表，但是在执行这个任务时使用+c 选项关闭了默认的自动提交选项(也可以在这个标志前后加上减号[-c-，效果是一样的]。在创建表 A 之后(请记住，没有提交这个操作)，创建另一个名为 B 的表，这一次也关闭了自动提交特性。然后对这两个表进行 Cartesian 联结，同样动态地关闭 DB2 CLP 的自动提交特性。最后，做一次回滚并再次运行同样的 SELECT 语句，这一次，这条语句失败了。

如果看看这个事务，就会发现没有执行提交操作。如果第一条 SELECT 语句没有包含+c 选项，那么就会提交创建表 A 和 B 的结果(因为这条 SELECT 语句成功了)；因此后面的回滚不会影响这两个已经提交的表，第二条 SELECT 语句会成功地返回与第一条 SELECT 语句相同的结果。

试一下相同的命令序列，但是这一次使用-c-选项。应该会看到同样的结果。在此之后，再试一次，这一次在第一条 SELECT 语句中不使用任何选项，看看第二条 SELECT 语句是否会返回结果。

操作系统可能对在一条语句中可以读取的最大字符数量有限制(即使命令行在显示器上转入下一行)。为了在输入长语句时解决这个限制，可以使用续行字符(\)。当 DB2 遇到续行字符时，会读取下一行并在处理时将两行合并。在两种 DB2 处理程序中都可以使用这个字符。但是，要知道 DB2 对一条语句的限制是 2MB(这对于命令行应该足够了)。图 4-23 演示了续行字符在 DB2 CLP 中的使用方法。

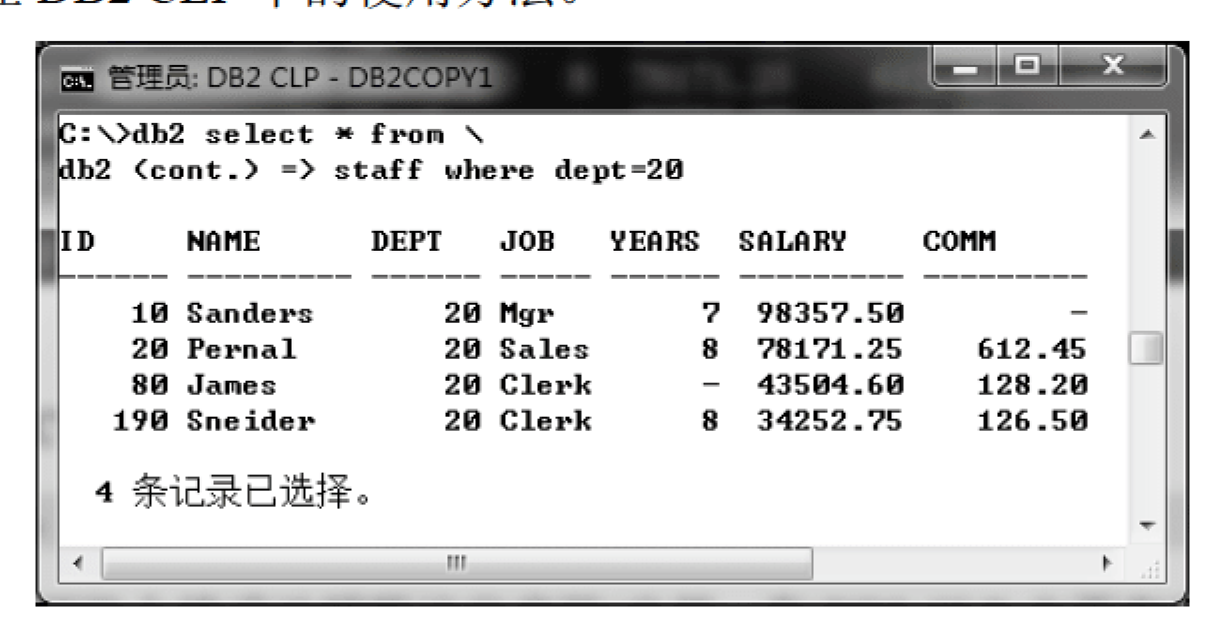


图 4-23　在 DB2 CLP 中使用续行字符

如果使用 DB2 CW 输入命令，那么下面这些特殊字符会导致问题：

```
$ & *() ; < > ? \ ' "
```

操作系统 shell 可能会错误地解释这些字符(当然，在 DB2 CLP 中不存在这个问题，因为它是为 DB2 命令专门设计的应用程序)。

可以将整个语句或命令放在引号中，从而表示希望由 DB2 解释系统操作符，而不是由操作系统进行解释，如下所示：

```
db2 "select * from staff where dept > 10"
```

试着在 DB2 CW 中输入相同的命令，但是不加引号，会发生什么？查看发出这个命令时所在目录的内容，一定会找到名为 10 的文件，其中包含 SQL 错误。为什么呢？DB2 解释 SQL 语句

```
select * from staff where dept
```

并将产生的内容放进文件 10。“>”符号是操作系统指令，表示将来自标准显示的输出管道连接到指定的文件(在这个示例中是文件 10)。select * from staff where dept 语句当然是不完整的 SQL 语句，因此会产生错误。不正确的结果是由于操作系统错误地解释了特殊字符。

查看 DB2 命令的帮助信息，如果想在 CLP 中查看命令的帮助信息，可以输入 db2 ?选项，如图 4-24 所示。

```
db2 ?
db2 ? command string
db2 ? SQLnnnn      (nnnn = 4 or 5 digit SQLCODE)
db2 ? nnnnn        (nnnnn = 5 digit SQLSTATE)
```

图 4-24　查看 DB2 命令的帮助信息

4.3.4　设置 DB2_CLPPROMPT 以定制 DB2 CLP

DB2 提供了两种从命令行界面输入命令的方式。当以交互(Interactive)方式使用 DB2 命令行处理器(DB2 Command Line Processor，DB2 CLP)时，不必在 DB2 命令或 SQL 查询前加上关键字 DB2。

请看一下图 4-25，在运行于交互方式下的 DB2 UDB CLP 中输入了 SELECT*... 语句。你知道这个特定表(STAFF)位于哪个数据库或实例吗？大概不知道吧(虽然对于这个特例，你可以猜测)；可是，DB2 知道！

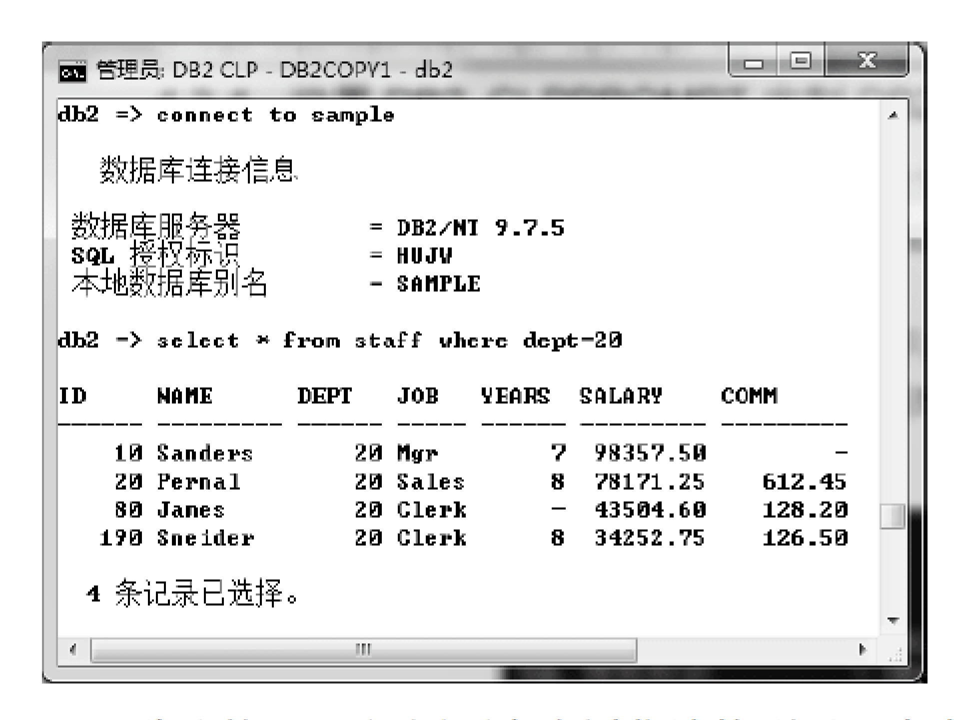


图 4-25　默认的 CLP 视图不会告诉你连接到了哪个实例

现在看一下图 4-26 中一模一样的查询。现在能回答前面的问题了吗？注意到有什么不同了吗？

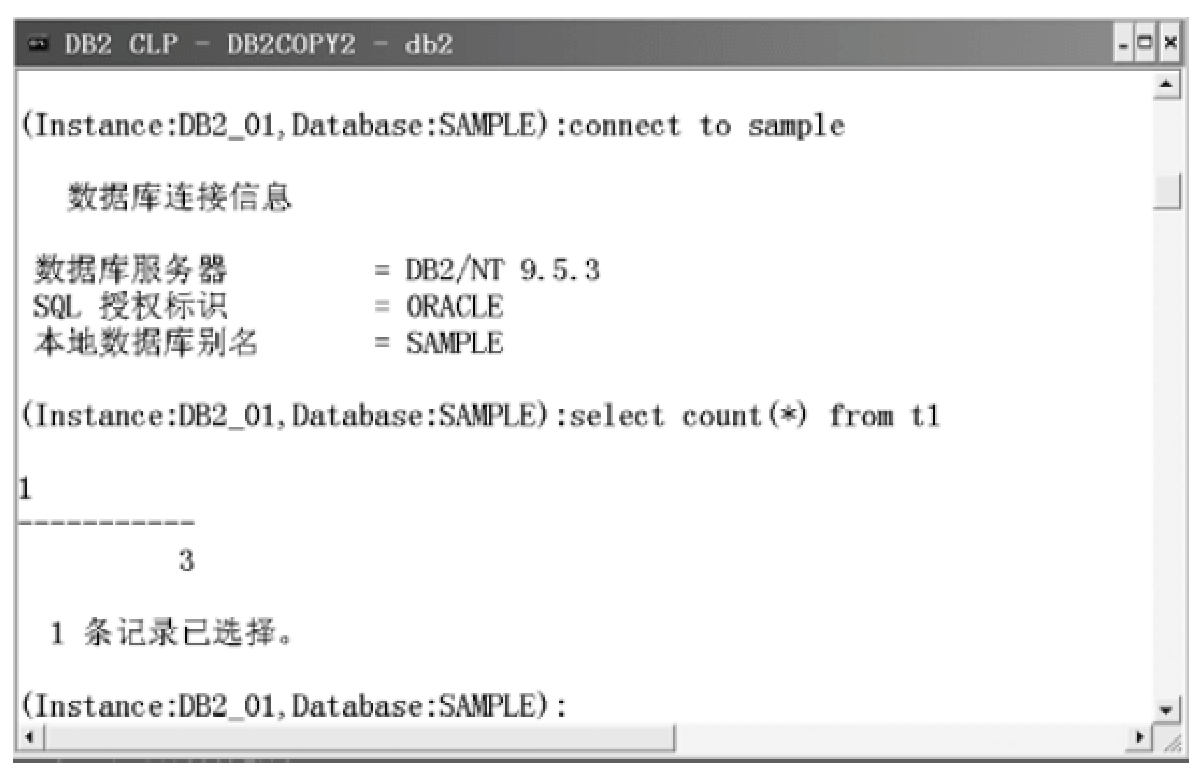


图 4-26　定制 CLP 以显示连接的实例和数据库

DB2 通过设置注册变量可以让你在运行于交互方式时，定制 DB2 UDB CLP 提示符(后面讲到的 DB2 CLP 就是运行于交互方式的 DB2 CLP)。

可以使用这项新的定制功能，把自己的文本和反映当前实例连接(instance attachment)和/或数据库连接(database connection)的上下文(context)变量添加到输出显示。下面介绍这项新的定制功能。

定制交互方式下的 DB2 CLP

现在，DB2 UDB CLP 提示符可以反映当前实例连接和数据库连接的上下文，还可以显示特定的字符消息。如果没有该项功能，运行位于交互方式下的 DB2 CLP 会显示硬编码的提示符，如图 4-27 所示。

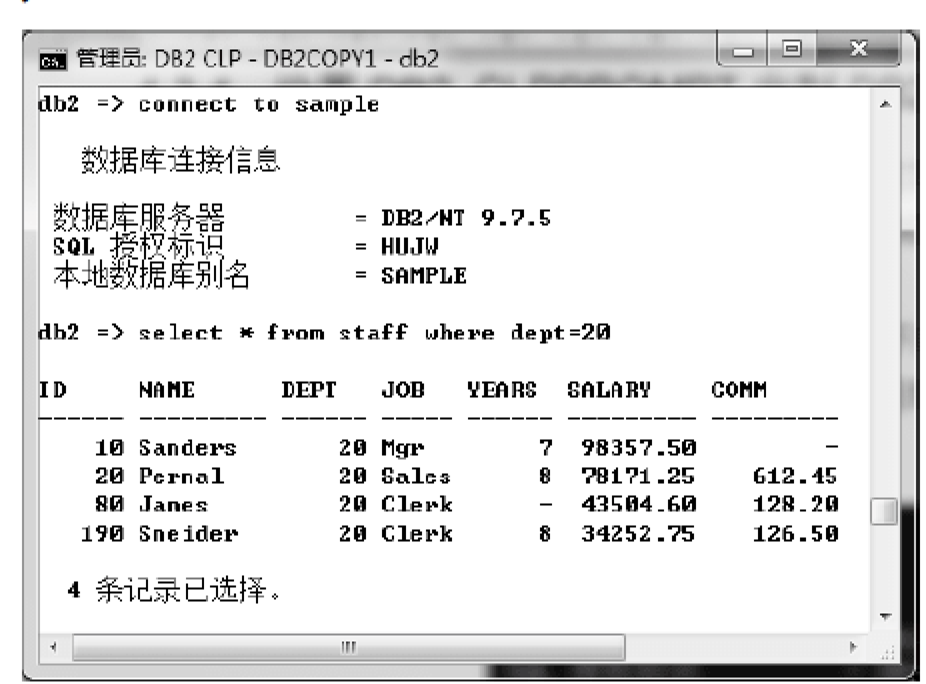


图 4-27　CLP 中的硬编码提示符

设置 DB2_CLPPROMPT 注册变量

要定制 DB2 CLP 命令提示符，请使用新的 DB2 注册变量——DB2_CLPPROMPT。

可使用 db2set 命令更新 DB2 注册变量(关于 DB2 注册变量和 db2set 命令的详细信息，已在第 2 章中讲解过了)，这些信息被立即存储到实例注册变量中。DB2 实例注册变量将这些更新过的信息应用到在进行更改之后启动的 DB2 服务器实例和应用程序。如果需要永久性设置某个环境变量，那么应该使用 db2set 命令在 DB2 服务器的启动实例注册变量内设置。而 db2set 命令则将环境变量永久性地设置在 DB2 实例注册变量中。

要查看受支持的全部注册变量的列表，请输入以下命令：

```
db2set -lr
```

要更改 DB2 UDB 注册变量的值，请输入以下命令：

```
db2set registry_variable_name=new_value
```

要查看被设置的全部 DB2 注册变量的列表，请输入以下命令：

```
db2set -all
```

可以将 DB2_CLPPROMPT 设置为长度不超过 100 个字符的任何文本字符串。这个定制的字符串可包含在运行时可替换的可选标记。如果这个注册变量在 DB2 CLP 会话期间发生更改，那么新的值在用户退出并且再次重新进入处理器后方可生效。

可以将 DB2 UDB CLP 定制为只显示一行字符串，这是最基本的形式。图 4-28 和图 4-29 演示了 DB2_CLPPROMPT 注册变量的设置以及 DB2 UDB CLP 的后续调用。

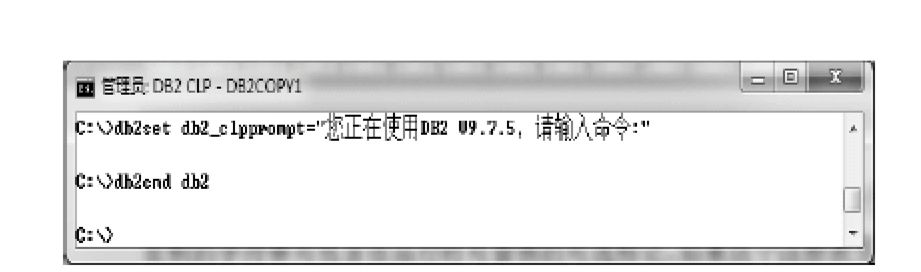


图 4-28　设置命令行提示符注册变量

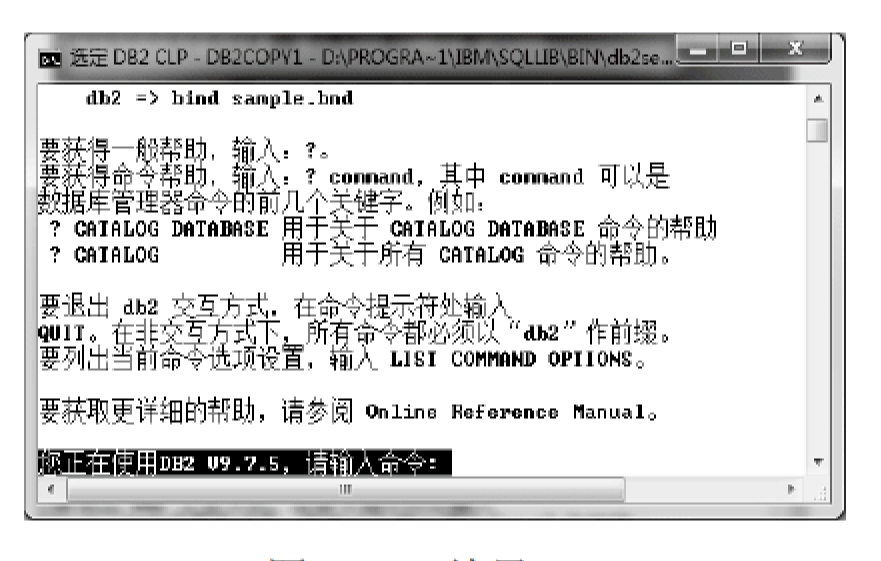


图 4-29　结果

现在，尽管这个示例挺有意思，但没什么大用。DB2_CLPPROMPT 注册变量可以附带一些关联变量，可以用这些变量向 DB2 CLP 返回一些关于当前的或默认的实例连接，以及当前连接的(或默认的)数据库的信息。

受支持的变量如表 4-1 所示。

表 4-1　受 CLP 支持的运行时变量

变　量	运 行 时 值
%ia	如果存在实例连接，为当前实例连接的授权标识(authid)；否则为空字符串
%i	如果存在实例连接，为当前连接的实例的本地别名；如果不存在本地实例连接，为 DB2INSTANCE 或 DB2INSTDEF 注册变量的值；否则为空字符串
%da	如果存在数据库连接，为当前数据库连接的授权标识；否则为空字符串
%d	如果数据库连接存在，为当前连接的数据库的本地别名；否则为 DB2DBDFT 注册变量值；再不然则为空值
%n	换行符

例如，要设置 DB2 UDB CLP 提示符，使其解析为：

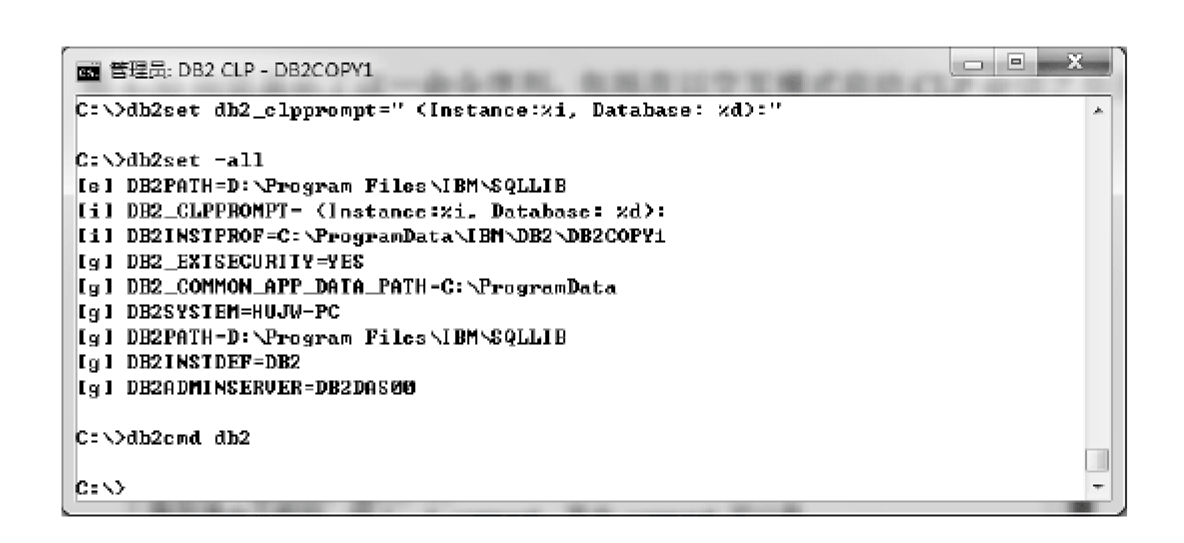
```
(Instance <instance_name>, Database <database name>):
```

输入以下命令：

```
db2set db2_clpprompt=" (Instance:%i, Database: %d):"
```

可以输入 db2set -all 命令来验证 DB2 UDB 概要注册表中的该项设置。

图 4-30 向你显示了这一命令序列，包括在以交互方式启动 CLP 会话之后的显示结果。



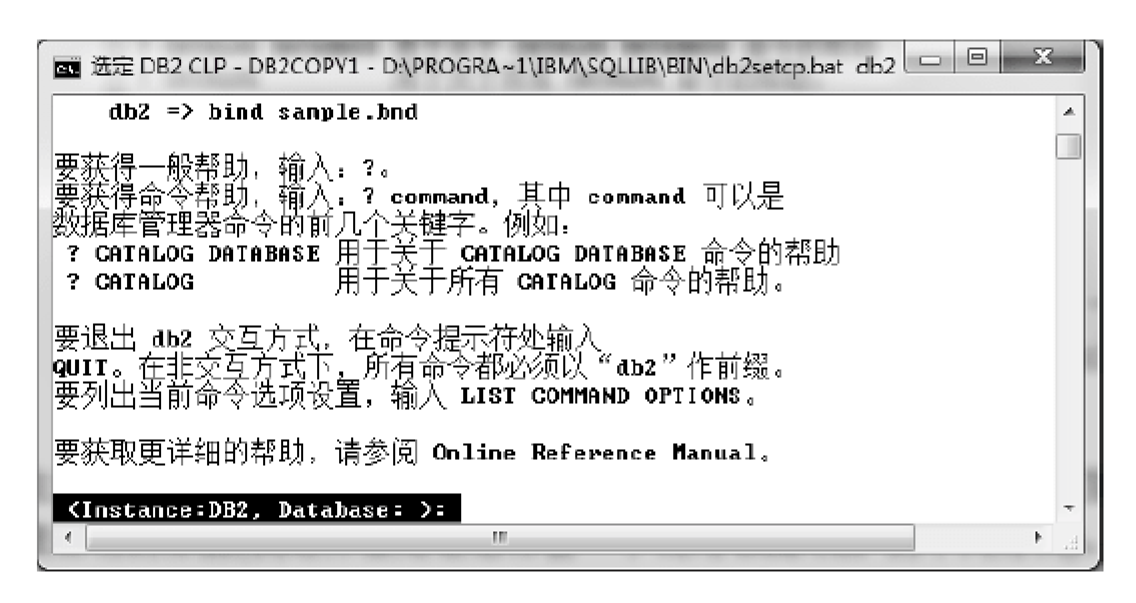


图 4-30　设置 DB2_CLPPROMPT 并验证其使用

请注意此例中，使用了未处于交互方式的 DB2 CLP，这就是为什么交互方式在同一窗口中被启动的原因(这样做是为了让你体会一下调用 DB2 CLP 的不同方式)。

在图 4-30 中，可以看到<*database_name*>变量没有值。该变量之所以为空，是因为在笔者的环境中没有数据库连接或者定义的默认数据库。

如果连接到数据库，这个变量就会更新，如图 4-31 所示。

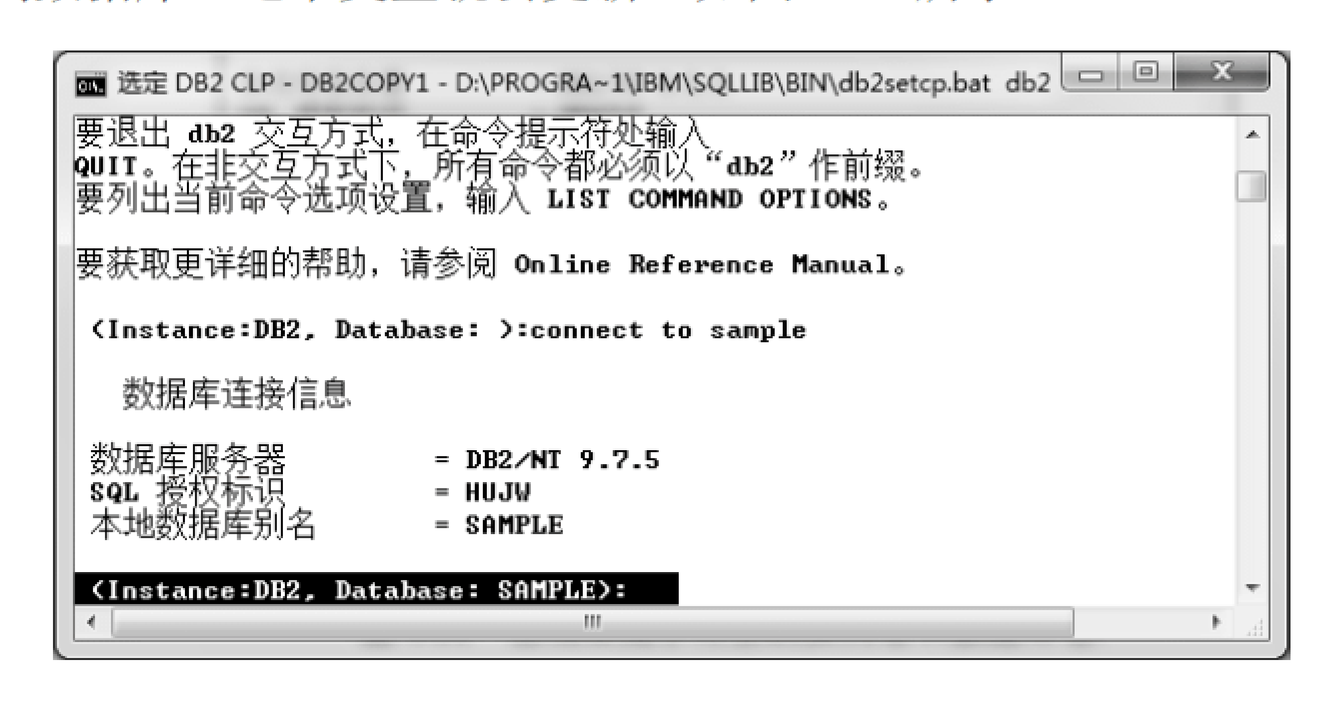


图 4-31　出现在提示符中的实例名和数据库名

如果从这个数据库断开连接，这个定制的字符串会反映出这一操作，如图 4-32 所示。

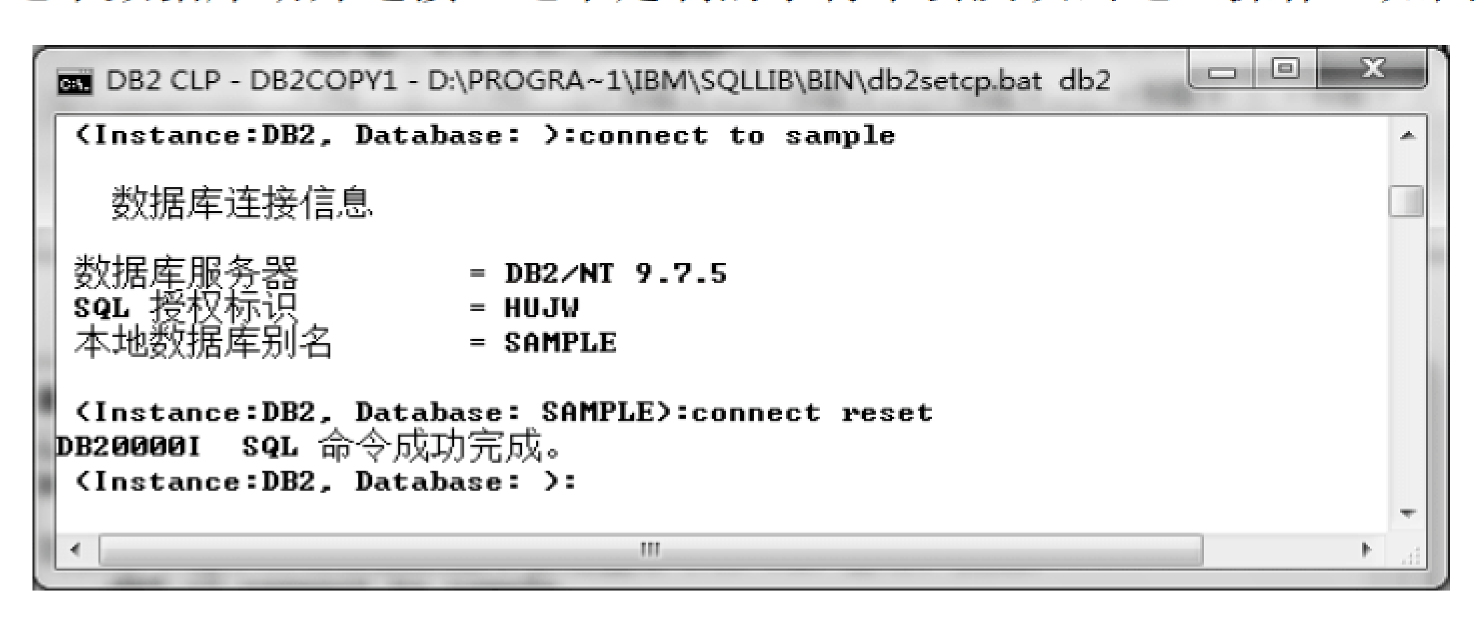


图 4-32　断开数据库连接，CLP 动态进行更新

4.4　配置 DB2 服务器的 TCP/IP 通信

客户端要想访问 DB2 数据库服务器，就必须先配置 DB2 服务器的通信协议，DB2 服务器才会接受来自远程 DB2 客户机的连接建立请求。

在配置 DB2 实例的 TCP/IP 通信之前，必须检查以下内容：

- 如果 DB2 服务器正在使用 TCP/IP，那么 DB2 客户机也必须正在使用 TCP/IP 才能建立连接。
- 标识“连接服务名称”和“连接端口”，或仅标识“连接端口”。

连接服务名称和连接端口

连接服务名称用于更新服务器上数据库管理器配置文件中的“服务名称”(svcename)参数。当指定“连接服务名称”时，必须以相同的“服务名称”、端口号和协议更新 services 文件，services 文件包含在服务器上定义的服务及其端口号。“服务名称”是任意的，但是在 services 文件内必须唯一。服务名称的样本值可以是 server1。“连接端口”在 services 文件中也必须唯一。端口号和协议的样本值可以是 50000/tcp。

连接端口

可以选择不使用“连接服务名称”而只是使用“连接端口号”更新服务器上数据库管理器配置文件中的“服务名称”(svcename)参数。这时就不会用到 services 文件，自然也不必更新 services 文件。如果正在使用分区格式的“DB2 企业服务器版”，那么必须确保端口号与“快速通信管理器”(FCM)或系统中的任何其他应用程序使用的端口号没有冲突。端口号的样本值可以是 50000。

要配置 DB2 实例的 TCP/IP 服务器通信，需要以下几个步骤。

4.4.1 在服务器上更新 services 文件

TCP/IP services 文件指定服务器应用程序侦听客户机请求的端口。如果在 DBM 配置文件的 svcename 字段中指定了服务名称，那么必须在 services 文件中添加一行，写入服务名称与端口号/协议的映射关系。如果在 DBM 配置文件的 svcename 字段中指定的不是服务名称而直接是端口号，那么不需要更新 services 文件。

在这里需要指出，services 文件的默认位置取决于操作系统，参考表 4-2。

表 4-2 services 文件的位置

操 作 系 统	目 录
Windows	%SystemRoot%\system32\drivers\etc，其中%SystemRoot%是系统定义的环境变量
Linux 或 UNIX	/etc

使用文本编辑器将“连接”条目添加至 services 文件。例如：

```
db2c_db2inst1  50000/tcp #DB2 连接服务端口
```

其中：

- db2c_db2inst1 表示连接服务名称。
- 50000 表示连接端口号(#50000 是 DB2 实例的默认端口)，读者可以根据自己需要更改。
- tcp 表示使用的通信协议。

注意：

这个步骤在创建数据库实例的时候会自动更新 services 文件，在 Windows 平台上，5000 是 DB2 实例的默认端口，而在 Linux/UNIX 平台上，60000 是 DB2 实例的默认端口，可以根据自己的需要更改成其他的端口号。

在 UNIX 平台上，在对应的/etc/services 文件中，默认配置为：

```
DB2 db2inst1      60000/tcp
DB2 db2inst1 1     60001/tcp
DB2 db2inst1 2     60002/tcp
DB2_db2inst1_END  60003/tcp
```

4.4.2　在服务器上更新数据库管理器配置文件

更新数据库管理器配置文件在配置 DB2 实例的 TCP/IP 通信过程中是必不可少的一环。必须使用服务名称(svcename)参数更新数据库管理器配置文件。

要更新数据库管理器配置文件，必须完成以下操作：

(1) 作为具有“系统管理员”(SYSADM)权限的用户登录系统。

(2) 启动 DB2 命令行处理器(CLP)。

(3) 通过输入下列命令，用“服务名称”(svcename)参数更新数据库管理器配置文件：

```
db2 update database manager configuration using svcename
[service name|port number]
        db2stop
        db2start
```

其中：

- service_name 是 services 文件中保留的服务名称。
- port_number 是 service_name 的相应端口号或空闲的端口号(假设未保留 service_name)，如果正在指定服务名称，那么使用的 svcename 必须与在 services 文件中指定的“连接服务名称”相匹配。

这里需要注意，svcename 不能联机配置，必须停启实例后才能生效。

在停止并再次启动数据库管理器之后，查看数据库管理器配置文件以确保这些更改已经生效。通过输入下列命令，查看数据库管理器配置文件：

```
db2 get database manager configuration | find /i "svcename"
```

4.4.3　设置 DB2 服务器的通信协议

要执行此任务，需要 sysadm 权限。为 DB2 实例设置通信协议是为 DB2 实例配置 TCP/IP 或 SSL 通信的主要任务的一部分。

DB2COMM 注册变量允许设置当前 DB2 实例的通信协议。如果 DB2COMM 注册变量未定义或设置为空，那么启动数据库管理器时不会启动任何协议连接管理器。

可以使用下列其中一个关键字来设置 DB2COMM 注册变量：tcpip 启动 TCP/IP 支持，ssl 启动 SSL 支持。

要为实例设置通信协议，可从 DB2 命令窗口输入 db2set DB2COMM 命令：

```
db2set DB2COMM=tcpip
```

例如，要将数据库管理器设置为对 TCP/IP 通信协议启动连接管理器，输入以下命令：

```
db2set DB2COMM=tcpip
db2stop
db2start
```

4.4.4　查看服务器通信端口的状态

在执行完上面 3 个步骤后，在系统中输入 netstat 来查看通信端口的状态，如图 4-33 所示。通信端口必须处于“LISTENING”状态才能侦听来自客户端的 tcp 请求。

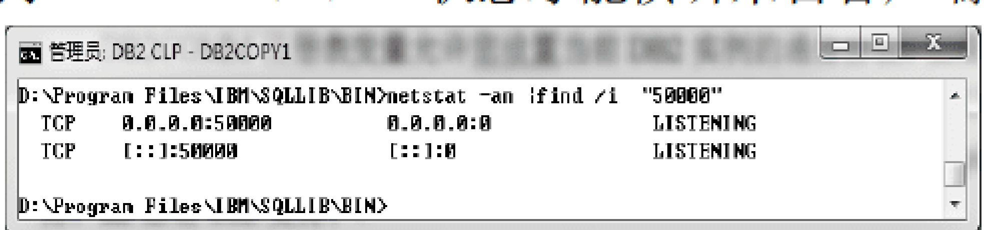


图 4-33　查看通信端口的状态

4.4.5　使用控制中心配置 DB2 服务器通信

控制中心的通信设置功能允许显示服务器实例在配置之后可使用的协议和配置参数，还允许修改已配置协议的参数值，以及添加或删除协议。

向服务器系统添加对新协议的支持时，通信设置功能检测并生成新协议的服务器实例参数值。在使用之前，可接受或修改这些值。当从服务器系统中除去对现存协议的支持时，

通信设置功能检测已除去的协议，并禁止服务器实例再次使用。

可添加尚未检测到的协议，但是在继续执行之前必须提供所有必需的参数值。“设置通信”窗口如图 4-34 所示。

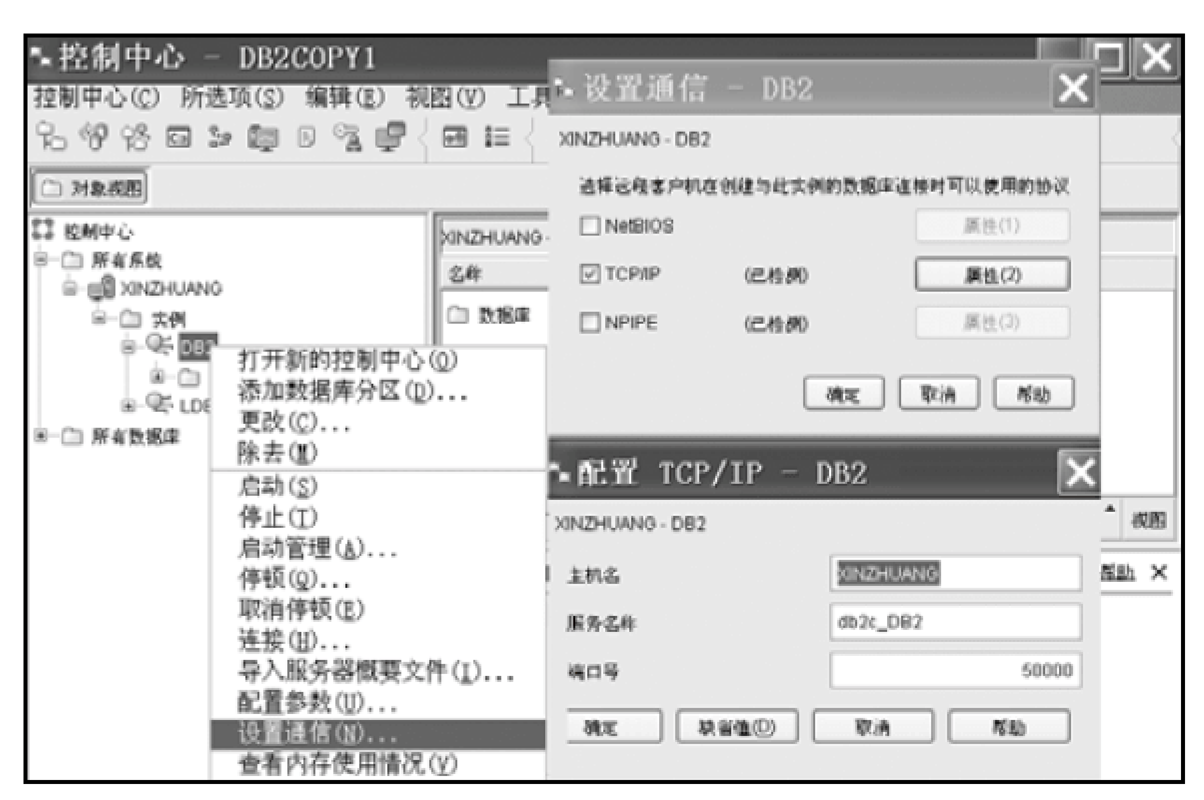


图 4-34　“设置通信”窗口

通信设置功能可用于维护本地和远程服务器实例的通信(要配置远程实例的 DB2 通信协议，必须要求远程 DB2 实例所在的服务器和本地服务器启动管理服务器(DAS))。

要修改先前已经配置的实例通信设置，可能需要更新客户机上的数据库连接目录。为此，可以：

- 在客户机上使用配置助手 CA。选择想要更改的数据库连接，在“所选项”菜单下选择“更改数据库”，这将启动“向导”以帮助进行更改。
- 根据服务器上已更改的值，在客户机上使用命令行处理器对节点取消编目和重新编目。

4.5　配置客户机至服务器通信

4.5.1　客户机至服务器通信概述

要想配置客户机至服务器通信，必须先了解客户机至服务器通信有关的基本组件：

- **客户机**——指的是通信的发起方。
- **服务器**——指的是来自客户机的通信请求的接收方。
- **通信协议**——指的是用来在客户机和服务器之间发送数据的协议。DB2 产品支持以下几个协议：

◊ TCP/IP。可根据版本进行更进一步的区分：TCP/IPv4 或 TCP/IPv6。
◊ IPC(进程间通信)。此协议用于本地连接。

客户机至服务器通信：连接类型

通常，提到设置客户机至服务器通信时指的是远程连接，而不是本地连接。

本地连接是数据库管理器实例与由那个实例管理的数据库之间的连接。换句话说，CONNECT 语句从数据库管理器实例发出给自己。本地连接是独特的，因为不需要设置通信并且使用了 IPC。

远程连接是在其中发出 CONNECT 语句到数据库的客户机和数据库服务器处于不同位置的连接。通常，客户机和服务器在不同的机器上。然而，如果客户机和服务器在不同的实例中，那么远程连接可能存在于同一台机器上。

另一个较不常用的连接类型是回送连接(loopback)。这是一种远程连接类型，该连接配置为从某个 DB2 实例(客户机)到相同的 DB2 实例(服务器)。

可以通过控制中心、CA 配置助手和 CLP 来配置客户机到服务器通信，下面分别讲解这 3 种不同的配置方式。

4.5.2 使用控制中心配置客户端通信

使用控制中心配置远程客户端和服务器通信，要求远程 DB2 实例所在的服务器和本地服务器必须启动管理服务器(DAS)，在远程和本地服务器上启动管理服务器后，执行下列步骤：

(1) 启动控制中心。

(2) 如果列出了包含想要的远程实例的系统，那么单击系统名称旁边的[+]号以显示“实例”文件夹。单击“实例”文件夹旁边的[+]以显示系统中所有实例的列表，然后转至步骤(13)。如果已列出包含想要的远程实例的系统，但需要的实例未出现在系统下面，那么转至步骤(8)。

(3) 如果未列出包含想要配置的远程实例的系统，那么选择**系统**文件夹，单击鼠标右键并选择**添加**选项。“添加系统”窗口将打开。

(4) 要向控制中心添加系统，可执行下列其中一项操作：

- 如果系统名称为空，那么单击 **Discover** 以显示网络上 TCP/IP 系统的列表。选择某个系统并单击**确定**。在“添加系统”窗口中填充系统信息。
- 如果已填写系统名称，那么单击 **Discover** 以调用已知 discovery。如果成功，那么在“添加系统”窗口中填充系统信息。

注意：
discovery 只对远程 TCP/IP 系统起作用。

(5) 单击**应用**以将系统添加至控制中心窗口。
(6) 单击**关闭**。
(7) 单击刚刚添加的系统名称旁边的[+]号以显示“实例”文件夹。
(8) 为新系统选择**实例**文件夹并单击鼠标右键。
(9) 选择**添加**选项。“添加实例”窗口将打开。
(10) 单击 **Discover** 以获取可用实例的列表并在系统中显示远程实例的列表。
(11) 选择想要添加的实例并单击**确定**。“添加实例”窗口将填充远程实例信息。
(12) 单击**关闭**。
(13) 选择要配置的实例并单击鼠标右键。
(14) 从弹出菜单中选择“设置通信”选项。“设置通信”窗口将打开。
(15) 使用“设置通信”窗口为该实例配置通信协议。
(16) 必须停止该实例，然后再重新启动，才可使这些更改生效：

- 要停止一个实例，选择该实例，单击鼠标右键并选择**停止**选项。
- 要启动一个实例，选择该实例，单击鼠标右键并选择**启动**选项。

4.5.3　使用 CA 配置客户机到服务器通信

客户机配置助手(CA)图形化工具能够非常快速、轻松地帮助我们配置客户端到服务器通信。在下面的这个案例中，我们使用 CA 对远程 DB2 服务器上的数据库进行编目。对数据库进行编目之后，就能够像访问本地数据库一样访问远程数据库。DB2 会在“幕后”执行所有通信过程。

配置步骤

(1) 熟悉以下远程数据库信息：
通信协议(PR) Protocol = TCPIP
IP 地址(IP)　IP Address or hostname = 192.168.1.1
端口号(PN)　　　Instance Port Number = 50000
数据库名称(DB)　　　Database Name = SAMPLE

提示：
在 Windows 中获取主机名的方法是在命令窗口中输入 *hostname*。
在 Windows 中获取 IP 地址的方法是在命令窗口中输入 *ipconfig*。

(2) 打开“配置助手”(可以通过“开始”菜单来访问)。

从“开始”菜单中找到 IBMDB2 的配置助手，单击启动，如图 4-35 所示。

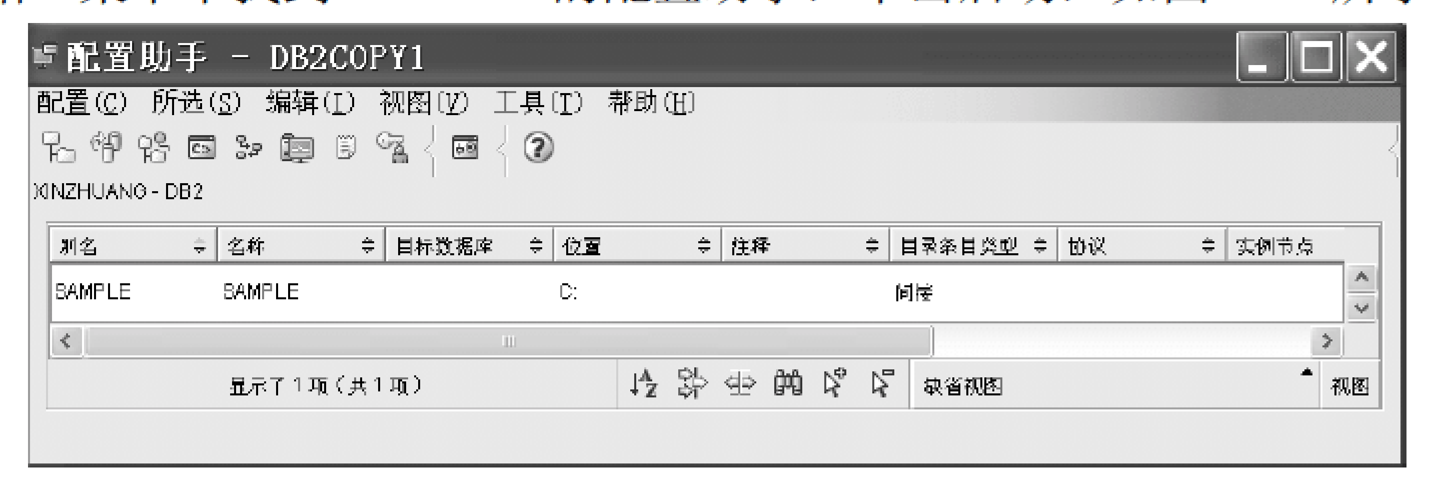


图 4-35　启动配置助手

(3) 打开“所选”菜单并选择“使用向导来添加数据库”子菜单。

(4) 在向导的“源”页面上，选择“手工配置与数据库的连接”选项，如图 4-36 所示。单击“下一步”按钮进入向导的下一页。

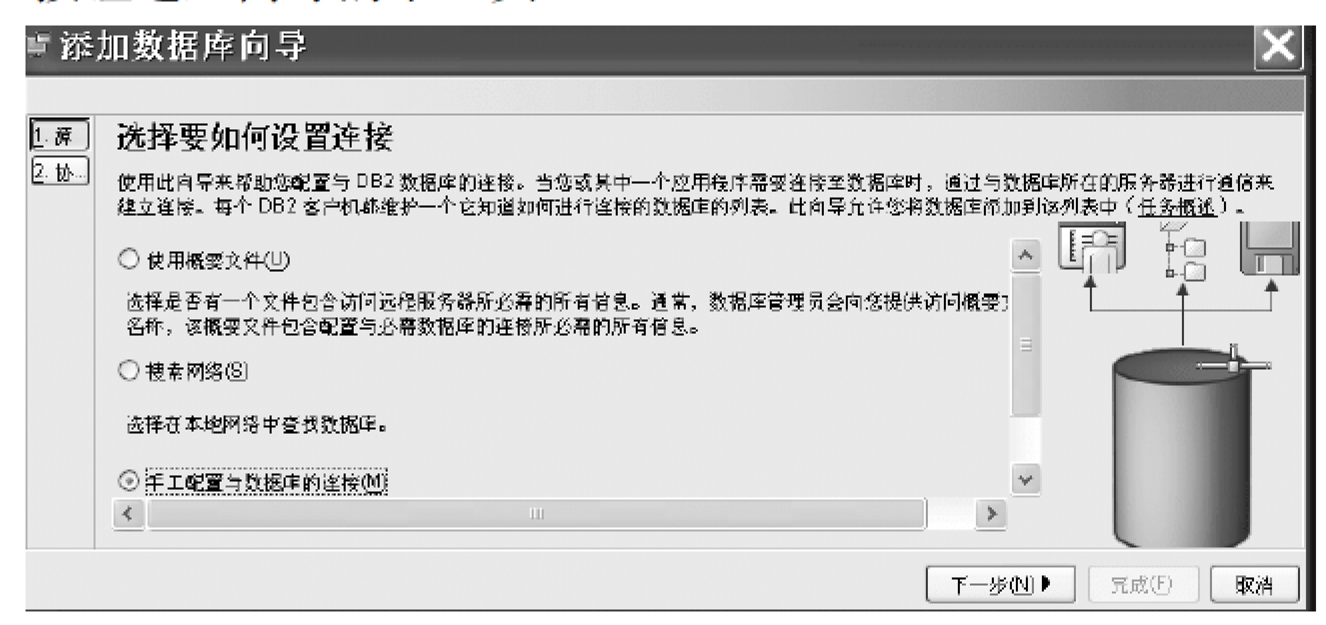


图 4-36　选择建立连接的方式

有 3 种连接方式：

- 使用概要文件：概要文件中包含访问远程服务器必需的所有信息(主机名、通信协议和端口)，适用于配置大批量客户端与服务器通信。
- 搜索(Discover)网络：如果使用这种自动配置，那么不需要提供任何详细的通信信息，就能够让 DB2 客户机与 DB2 服务器进行联系。适用于本地网络，可以自动搜索到实例和数据库，但是需要在服务器端配置并启动数据库管理服务器(DAS)；在大型网络上，搜索可能要花费很长时间。
- 手工配置与数据库的连接：需要提供主机名、通信协议和通信端口。

(5) 在向导的“协议”页面，选择 TCP/IP 选项，如图 4-37 所示。单击“下一步”按钮进入向导的下一页。



图 4-37　选择通信协议

(6) 在向导的 TCP/IP 页面，输入在第(1)步中记下来的主机名、IP 地址以及端口号，如图 4-38 所示。单击“下一步”按钮进入向导的下一页。



图 4-38　指定 TCP/IP 通信参数

注意：

如果在本地服务文件中有一个条目，其中定义了与远程服务器上实例监听的端口对应的端口号，那么可以使用“服务名称”(Service Name)选项。在使用这个选项时，DB2 会查看本地机器(而不是服务器)上的服务文件。如果要使用这个选项，就必须在这个文件中添加一个条目。

(7) 在向导的“数据库”页面的“数据库名称”框中，输入在第(1)步中记下来的远程服务器上的数据库名。注意，“数据库别名”(Database Alias)框会自动填上相同的值，如图 4-39 所示。数据库别名是本地应用程序用来连接这个数据库的名称。因为已经定义了名为 SAMPLE 的本地数据库，所以 DB2 不允许对同名的另一个数据库进行编目。因此，必须使用别名。对于这个示例，将数据库别名改为 SAMPLE1。如果愿意，可以输入可选的注释。单击“下一步”按钮进入向导的下一页。

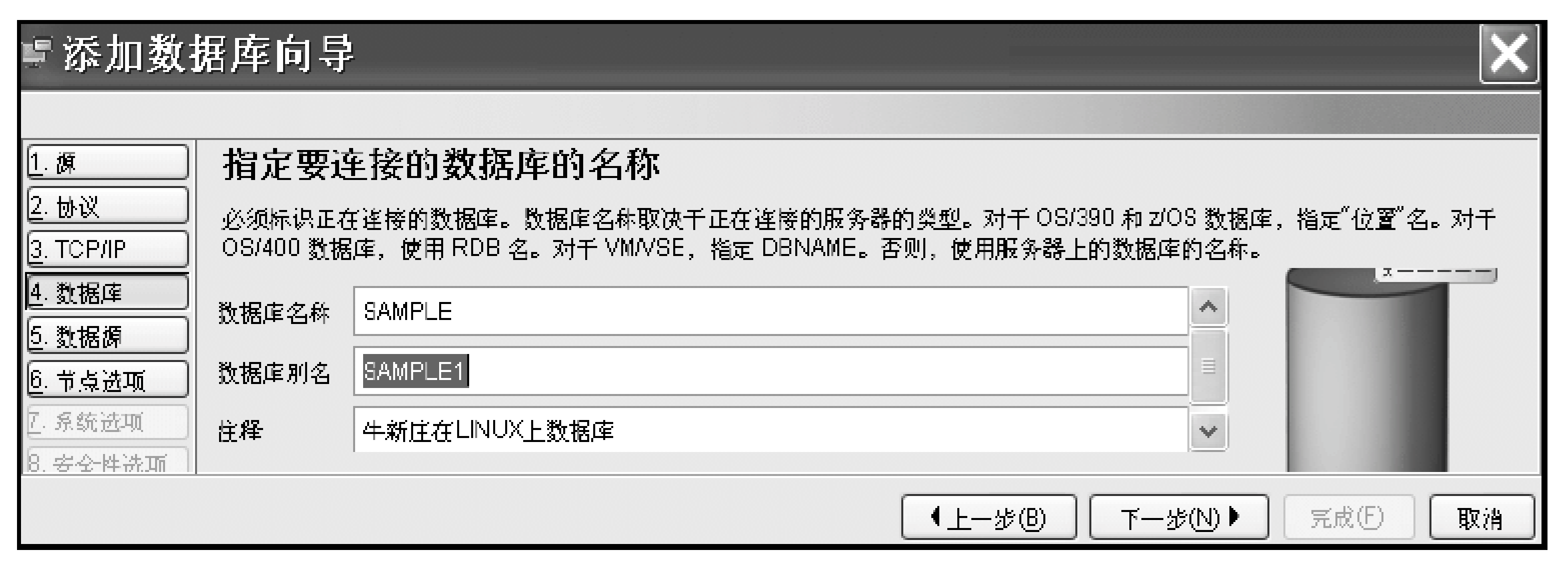


图 4-39　指定要连接的数据库名称

(8) 在向导的“数据源”(Data Source)页面，可以将这个新数据库(数据源)注册为 ODBC 数据源(这个步骤是可选的)。这会在 Windows ODBC Manager 中注册数据源。对于这个示例，因为不使用 ODBC，所以未选中“为 CLI/ODBC 注册此数据库”，如图 4-40 所示。单击 “下一步”按钮进入向导的下一页。

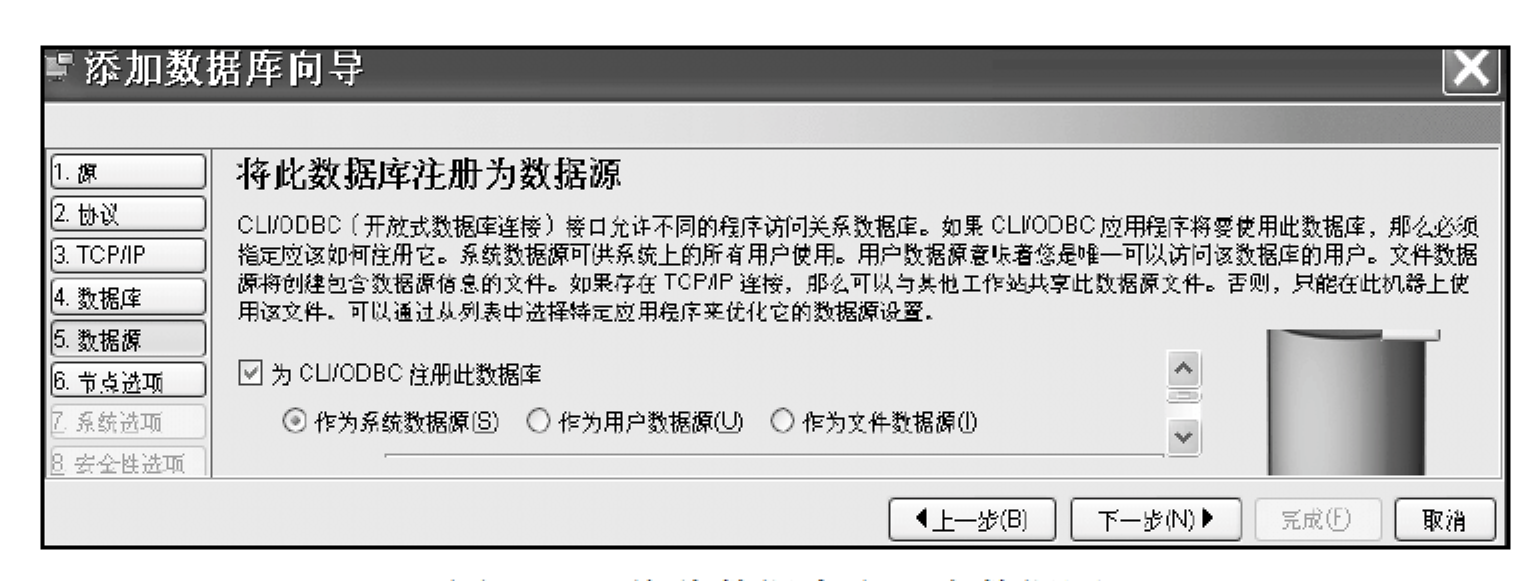


图 4-40　将此数据库注册为数据源

(9) 在向导的“节点选项”(Node Options)页面，指定远程数据库所在的服务器的操作系统。这个实验中的所有工作站都使用 Microsoft Windows。确保在下拉列表中选择 Windows。“实例名”框应该是 DB2。如果不是，就将值设置为 DB2，如图 4-41 所示。单击“下一步”按钮进入向导的下一页。

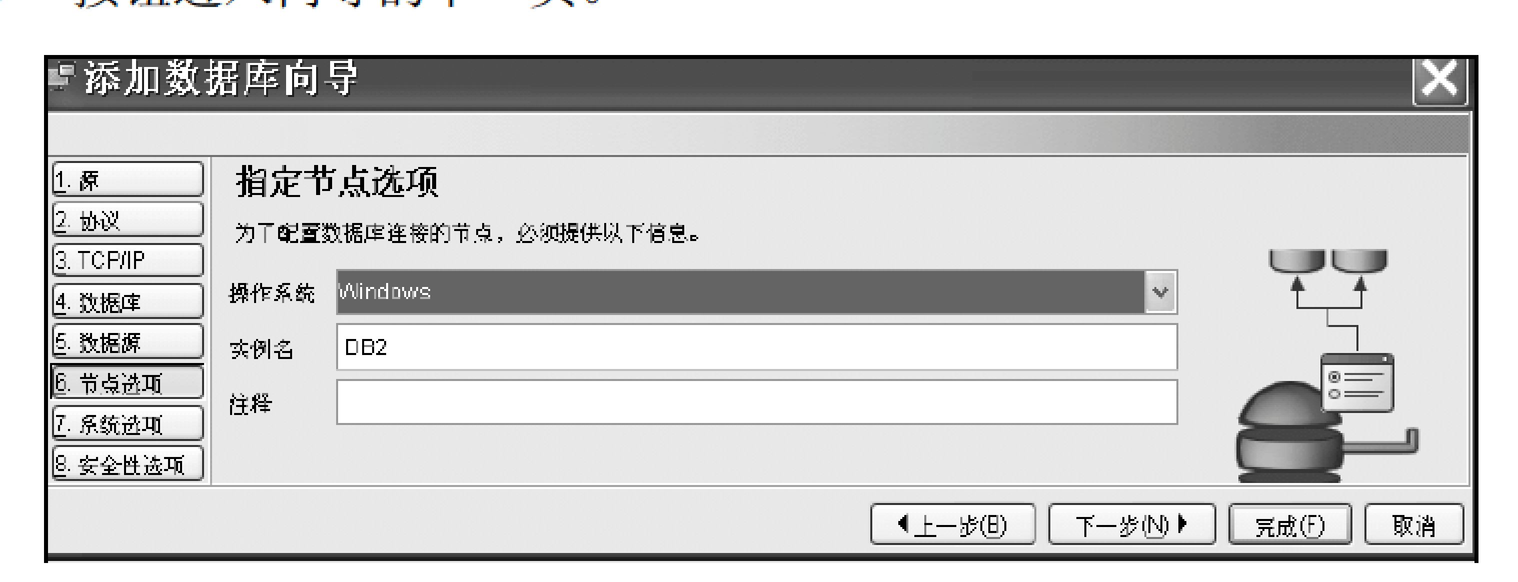


图 4-41　指定节点选项

(10) 在向导的“系统选项”(System Options)页面，检查系统和主机名是否正确，检查操作系统设置，如图 4-42 所示。单击“下一步”按钮进入向导的下一页。



图 4-42　指定系统选项

(11) 在向导的“安全性选项”页面，可以指定希望执行用户身份验证的位置和使用的身份验证方法，如图 4-43 所示。选择“使用服务器的"DBM 配置"中的认证值”选项，这将使用远程实例的配置文件中由 AUTHENTICATION 参数指定的方法。单击“完成”按钮，对远程数据库进行编目并关闭向导。这时应该会出现一个确认框，如图 4-44 所示。单击“测试连接”按钮，确认可以成功地连接数据库。另外，这也会确认由你提供的用户名和密码是远程服务器上定义的有效用户名和密码(因为服务器的 AUTHENTICATION 参数很可能设置为 SERVER)，如图 4-45 所示。如果连接测试成功，就会成功地对远程数据库进行编目。如果测试不成功，那么返回到向导并检查指定的值是否正确(单击“更改”按钮可返回到向导设置)。如果还想添加此远程服务器上的其他数据库，单击“添加”按钮继续添加数据库。



图 4-43　指定安全性选项

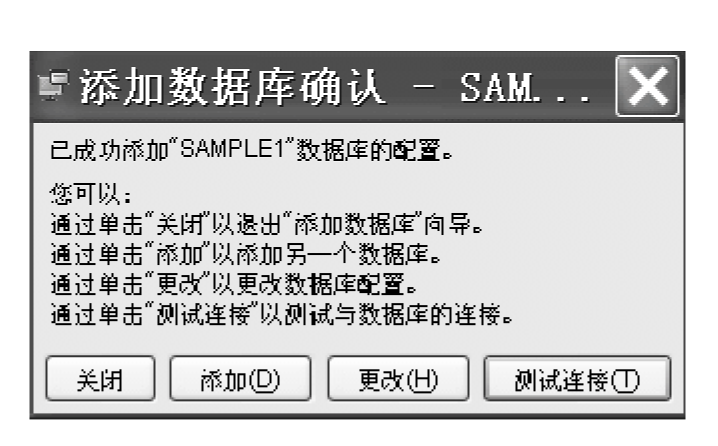


图 4-44　测试数据库连接

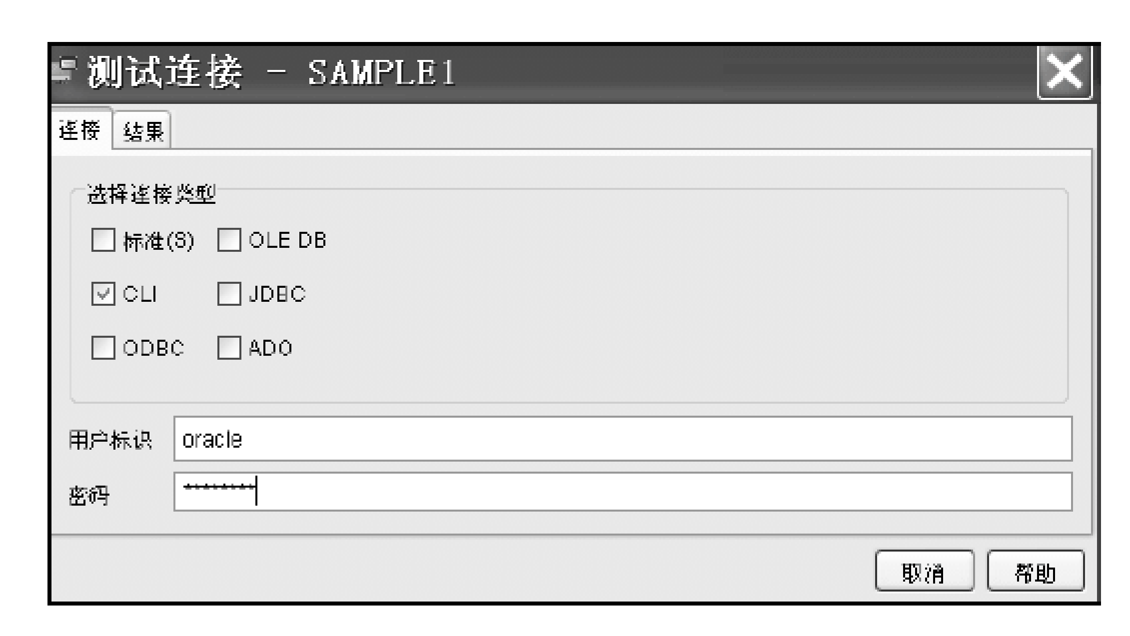


图 4-45　选择连接类型

注意：

根据需要选择测试的连接类型，一般可以选择 JDBC，输入正确的“用户标识”和“密码”进行测试。

(12) 测试连接正确后，可以打开控制中心并尝试查看刚才编目的远程数据库以及其中的数据库表，如图 4-46 所示。

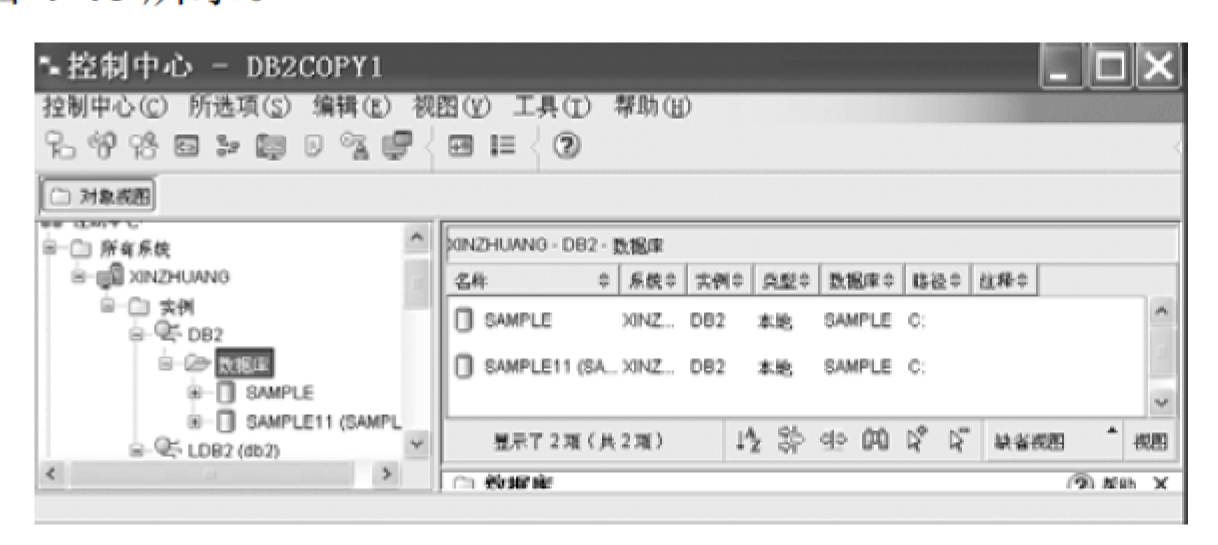


图 4-46　验证数据库编目是否正确

至此，我们已经使用配置助手配置好了客户机到服务器通信。

4.5.4　深入了解 DB2 节点目录、数据库目录

前面讲解了如何通过控制中心或 CA 来配置客户机到服务器通信，其实这两种方式本质上都是在后台通过调用命令来完成的。下面讲解如何使用命令行来配置客户机到服务器通信。在讲解客户机到服务器通信时，我们必须先弄清楚节点目录、系统数据库目录、本地数据库目录这几个概念。记得在刚开始学习 DB2 时，在成功地安装完之后，在配置客户机通信时，笔者被文档中的“cataloging nodes and databases”(编目节点和数据库)这件事给弄糊涂了。catalog 这个词与过去惹人喜爱的 SYSCAT 和 SYSIBM 目录相比有着动词化的意味。有时候，我会对着 DB2 大声埋怨：“我不想编目任何东西，我只是想在远程客户端

通过运行一条 SELECT 语句来确保已正确地安装了 DB2。”经过对节点目录和数据库目录概念的仔细研究，我了解到只有在创建数据库之后 DB2 才会有数据库目录；不需要在本地机器上将节点和数据库编目——只有在连接到服务器的客户机上才需要编目。

在 DB2 中，目录是存储有关系统、数据库及其连接信息的二进制文件。DB2 中有以下几种目录：

1. 节点目录

节点目录用于存储远程数据库的所有连通性信息。下面只介绍 TCP/IP 协议。在节点目录中，大多数项将和 TCP/IP 信息有关，比如机器(其中包含了想连接的数据库)的主机名或 IP 地址，还有相关的 DB2 实例的端口号。下面是一些与节点目录相关的命令：

要列示本地节点目录的内容，可使用 LIST NODE DIRECTORY 命令。请从 CLP 发出下面这个命令：

```
db2 list node directory
```

要将信息输入节点目录进行编目，请从 CLP 发出 catalog 命令：

```
db2  catalog TCPIP node <node name>  remote <hostname or IP address>
       server <port_name or port_number>
```

例如：

```
db2   catalog TCPIP node n1 remote 9.26.138.35    server 50000
```

要除去节点目录，请从 CLP 发出 uncatalog 命令：

```
db2  uncatalog node n1
```

为了得到想要连接的远程实例的端口号，可以通过查看远程实例的 dbm cfg 中的 svcename 参数来实现。该值通常对应于 TCP/IP services 文件中的某一项。

在每个数据库客户机上都创建并维护节点目录。对于具有客户机可以访问的一个或多个数据库的每个远程客户端，该目录都包含一个条目。无论何时请求数据库连接或实例连接，DB2 客户机都会使用该节点目录中的通信信息。该节点目录中的条目还包含客户机与远程实例通信时要使用的通信协议的类型信息。在图 4-47 中，如果想在 Workstation1 的 inst1 实例下访问同一台机器上的 inst2 实例和远程 Workstations2 上的 inst3 实例，那么必须在 inst1 下创建本地实例 inst2 和远程实例 inst3 的节点目录。

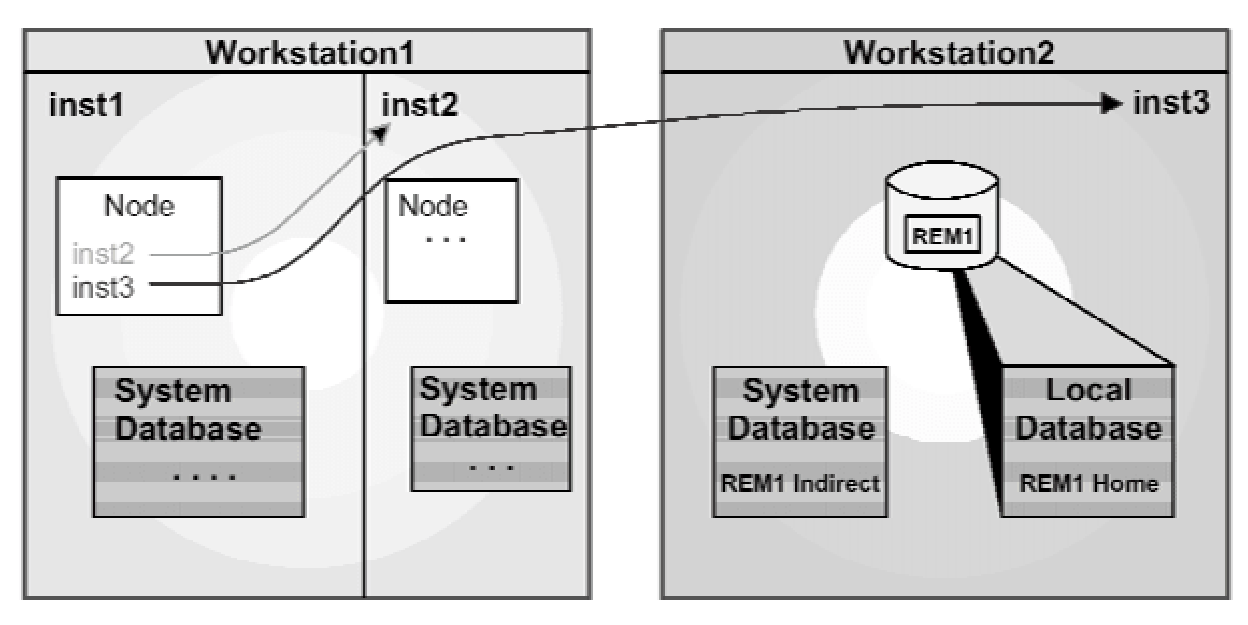


图 4-47　访问同台机器的另一实例和远程实例

可以这样理解，要读取某个表，就必须先访问数据库；可是要想访问数据库，就必须先访问实例，因为数据库是包含在实例中的。但是我们无法直接访问实例，因为实例不是“物理”的，而是逻辑的，实例是一组后端进程和共享内存的结合。所以在这种情况下，就需要为实例建立物理“映射”，这就是节点目录的由来，所以节点目录是和实例对应的。如果实例就在本地，那么在创建实例的时候，默认会创建和实例同名的本地目录。这是隐式的，反正本地访问也用不到这个节点目录。但是如果在客户机上需要访问远程实例，就必须为该实例建立和实例对应的节点目录。这个节点目录告诉我们该实例驻留在哪个机器上(IP 地址，主机名)，使用什么通信协议(设置 DB2COMM 变量)和使用的通信端口(SVCENAME)。

节点目录默认在实例目录下，有两个文件：SQLNODIR 和 SQLNOBAK。其中，SQLNOBAK 是 SQLNODIR 的备份，当 SQLNODIR 被损坏时，把 SQLNOBAK 改名为 SQLNODIR 即可。这个文件是二进制的，不过在 Windows 中通过编辑器可以看到其中一些可读信息。

2. 系统数据库目录(或系统 db 目录)

系统数据库目录包含本地数据库目录以及从远程映射到本地的数据库目录，是我们访问数据库的入口之一，连接数据库时首先去系统数据库目录中判断这个数据库是否存在，然后再判断这个数据库是本地数据库还是远程数据库。如果是本地数据库，就直接到本地物理目录上访问；如果是远程数据库，那么还要寻找这个远程数据库位于哪个节点上，然后再到节点目录中找到这个节点的通信信息。系统 db 目录是在实例级上进行存储的；对于数据库管理器的每个实例，都存在系统数据库目录文件，该文件对于针对此实例编目的每个数据库都包含条目。因此，如果打算删除实例，那么应当考虑备份其中的内容。

要列出系统 db 目录的内容，请从 CLP 发出下面这个命令：

```
$db2 list db directory
系统数据库目录
```

```
目录中的条目数目 = 2
数据库 1 条目:
数据库别名                = SAMPLE1
数据库名称                = SAMPLE
节点名                    = BJ141
注释                      = 牛新庄在 Linux 中的数据库
目录条目类型              = 远程
认证                      = SERVER_ENCRYPT
目录数据库分区号          = -1
备用服务器主机名          =
备用服务器端口号          =
数据库 2 条目:
数据库别名                = SAMPLE11
数据库名称                = SAMPLE
本地数据库目录            = C:
数据库发行版级别          = c.00
注释                      =
目录条目类型              = 间接(indirect)
目录数据库分区号          = 0
备用服务器主机名          =
备用服务器端口号          =
```

在上述命令输出中，任何包含单词“indirect”的项都意味着：该项适用于本地数据库(也就是驻留在当前正在使用的机器上的数据库)。该项还会指向由“Database drive”项(在 Windows 中)或“Local database directory”项(在 UNIX 中)指示的本地数据库目录。

任何包含单词“Remote”的项都意味着：该项适用于远程数据库(也就是驻留在其他机器上而非当前正在使用的机器上的数据库)。该项还会指向由“Node name”项指示的节点目录。

要将信息输入系统 db 目录，需要使用 catalog 命令：

```
db2 catalog db<db_name>  as <alias> at node  <nodename>
```

例如：

```
db2 catalog db   mydb   as yourdb  at node  mynode
```

节点名是指向节点目录中某一项的指针。在发出这条命令之前该项必须已经存在。

要除去数据库目录，请从 CLP 发出 uncatalog 命令：

```
db2  uncatalog db sample1
```

通常只有在将信息添加到远程数据库的系统 db 目录时才使用 catalog 命令。对于本地数据库来说，当发出 CREATE DATABASE 命令创建数据库之后就自动创建 Catalog 项，将隐式地对数据库进行编目。

系统数据库目录中包含以下内容：

- 数据库名称、别名和注释
- 本地数据库目录的位置
- 目录条目类型(remote 表示数据库在远程数据库，indirect 表示是本地数据库)
- 节点名(此节点名和节点目录中的节点名匹配)

系统数据库目录默认在实例目录下，有 3 个文件：SQLDBDIR、SQLDBBAK 和 SQLDBINS。其中，SQLDBBAK 是 SQLDBDIR 的备份，当 SQLDBDIR 被损坏时，把 SQLDBBAK 改名为 SQLDBDIR 即可。文件 sqldbins 只有分区数据库才会用到，是指向共享文件系统中另一个文件的符号链接。这些文件是二进制的，不过在 Windows 中，通过编辑器可以看到其中一些可读信息。

本地数据库目录(或本地 db 目录)

本地数据库目录包含了有关本地数据库(也就是驻留在目前正在使用的机器上的数据库)的信息。本地数据库目录驻留在数据库内部。当使用 create database 命令创建数据库时，将隐式地对数据库进行编目。在该目录中会添加一项。

要列出本地数据库目录的内容，请发出以下命令：

```
$db2 list db directory on c:
数据库 1 条目:

 数据库别名                        = BANK
 数据库名称                        = BANK
 数据库目录                        = SQL00002
 数据库发行版级别                  = c.00
注释                              = 信贷 12 级分类数据库
 目录条目类型                      = 本地
 目录数据库分区号                  = 0
 数据库分区号                      = 0
```

其中，可以从系统 db 目录相应项中的“Database drive”项(Windows 中)或“Local database directory” 项(UNIX 中)获取<path>。

节点目录、系统数据库目录和本地数据库目录之间的关系，如图 4-48 所示。

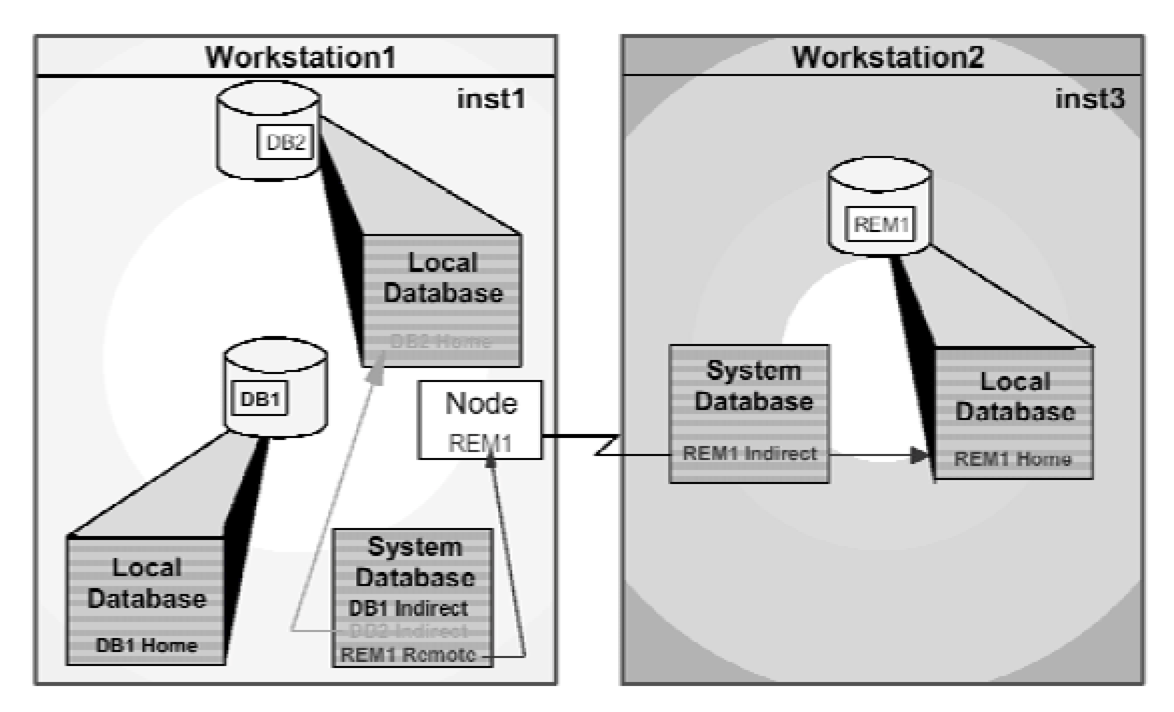
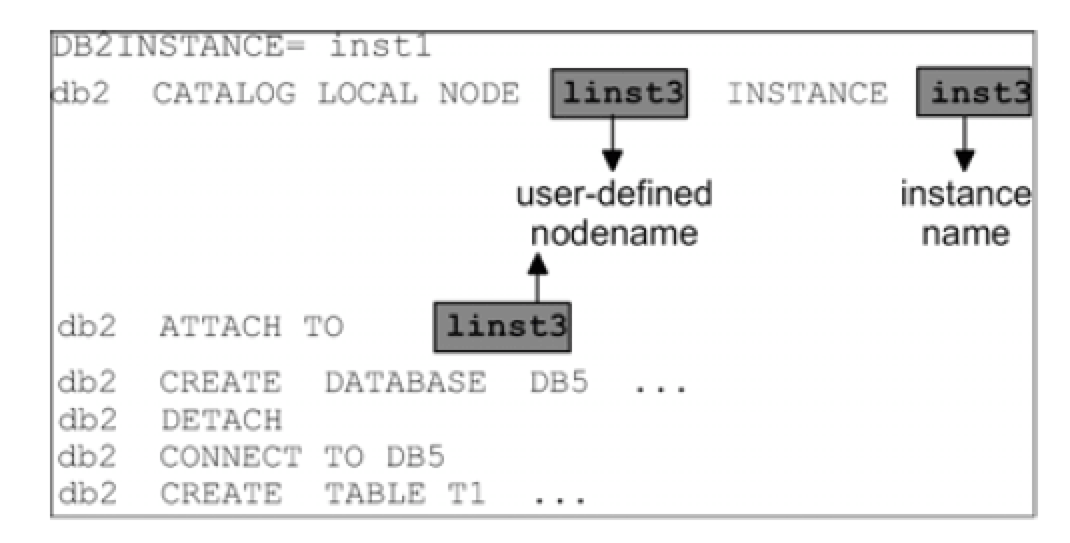


图 4-48　节点目录、系统数据库目录和本地数据库目录之间的关系

如果在 Workstation2 连接 inst3 实例：



如果在 Workstation1 连接远程实例和远程数据库：

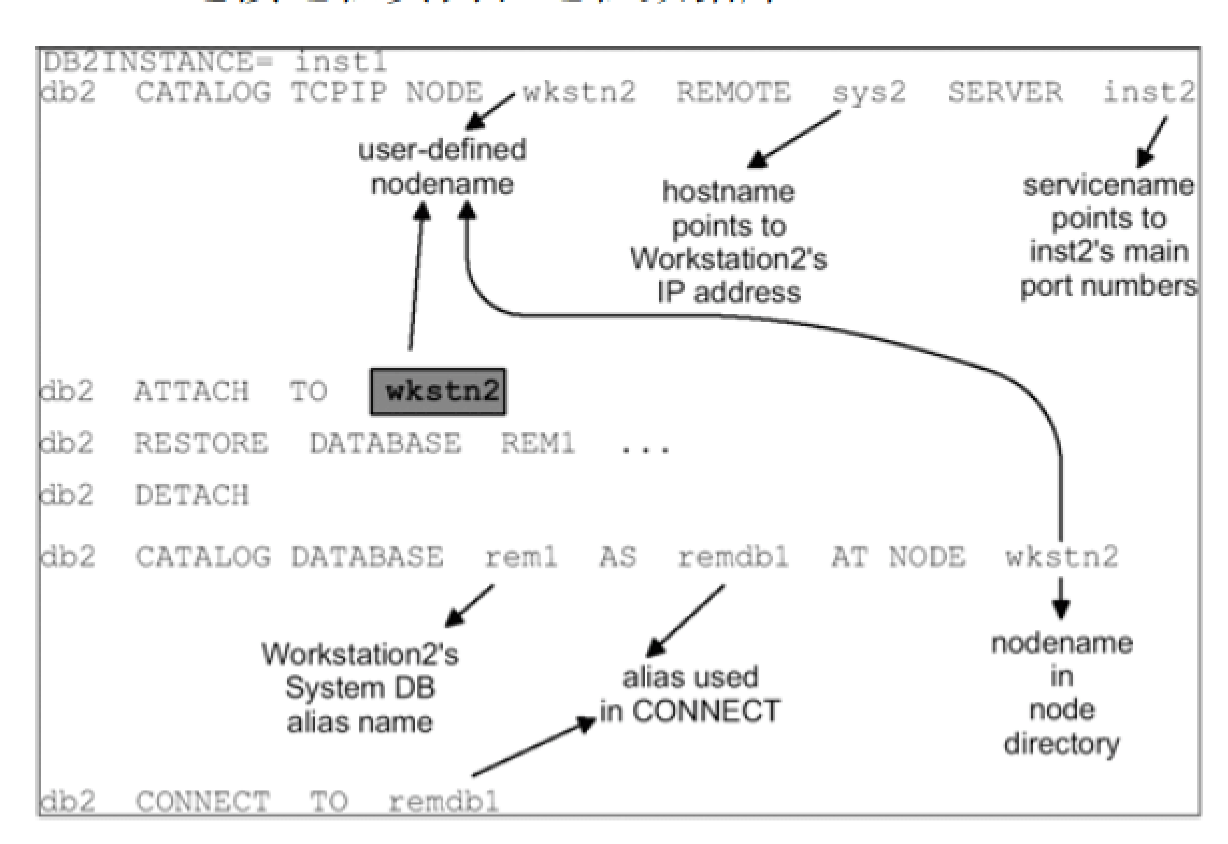


图 4-49 完整地总结了节点目录、系统数据库目录和本地数据库目录之间的关系。

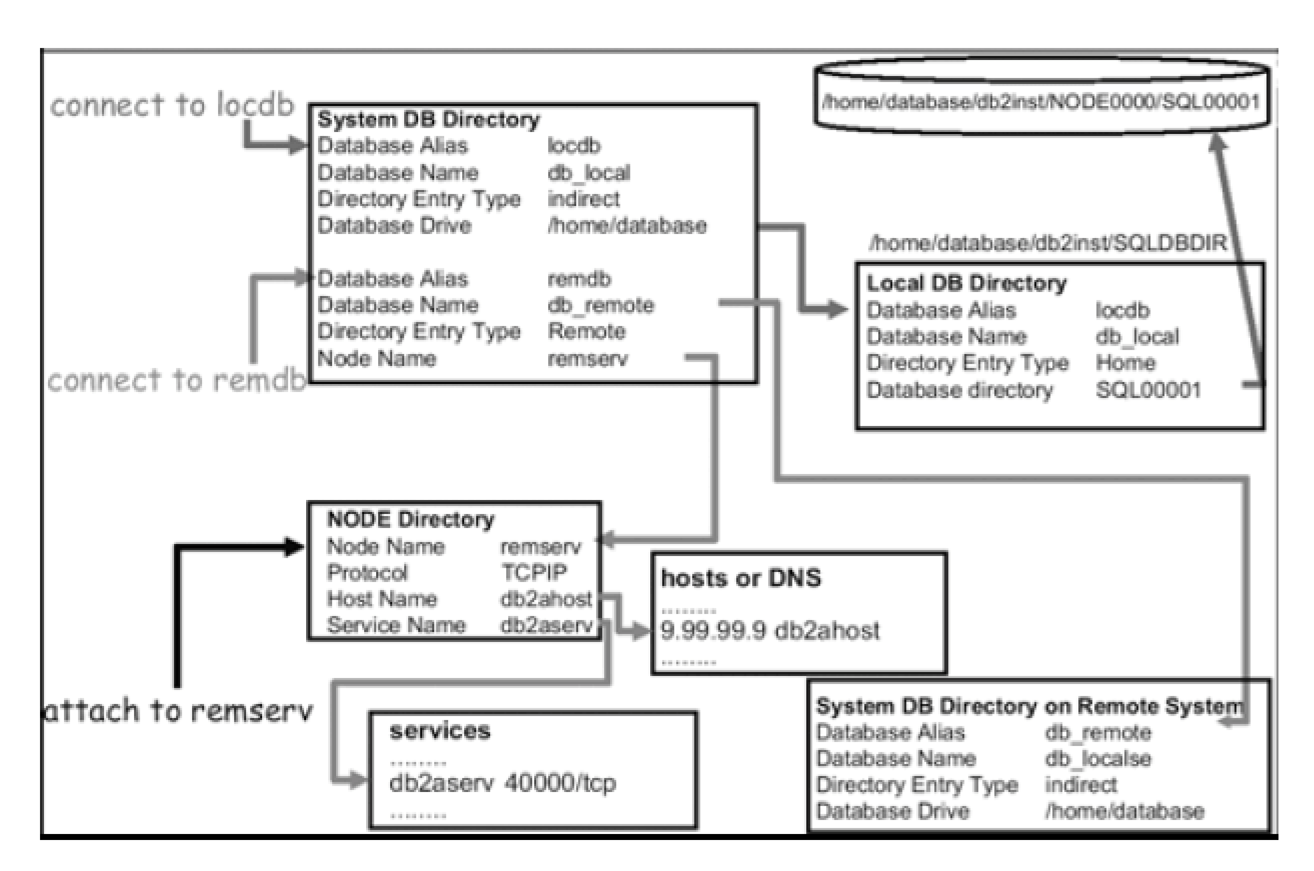


图 4-49 节点目录、系统数据库目录和本地数据库目录之间关系的总结

4.5.5 使用 CLP 配置客户机到服务器通信的案例

在了解了节点目录、系统数据库目录和本地数据库目录之后，如果想配置客户端到服务器端的通信，那么可以参考下面的图 4-50，其中归纳了客户端到服务器端通信的所有配置参数，读者可以根据自己的实际情况填写这张表。

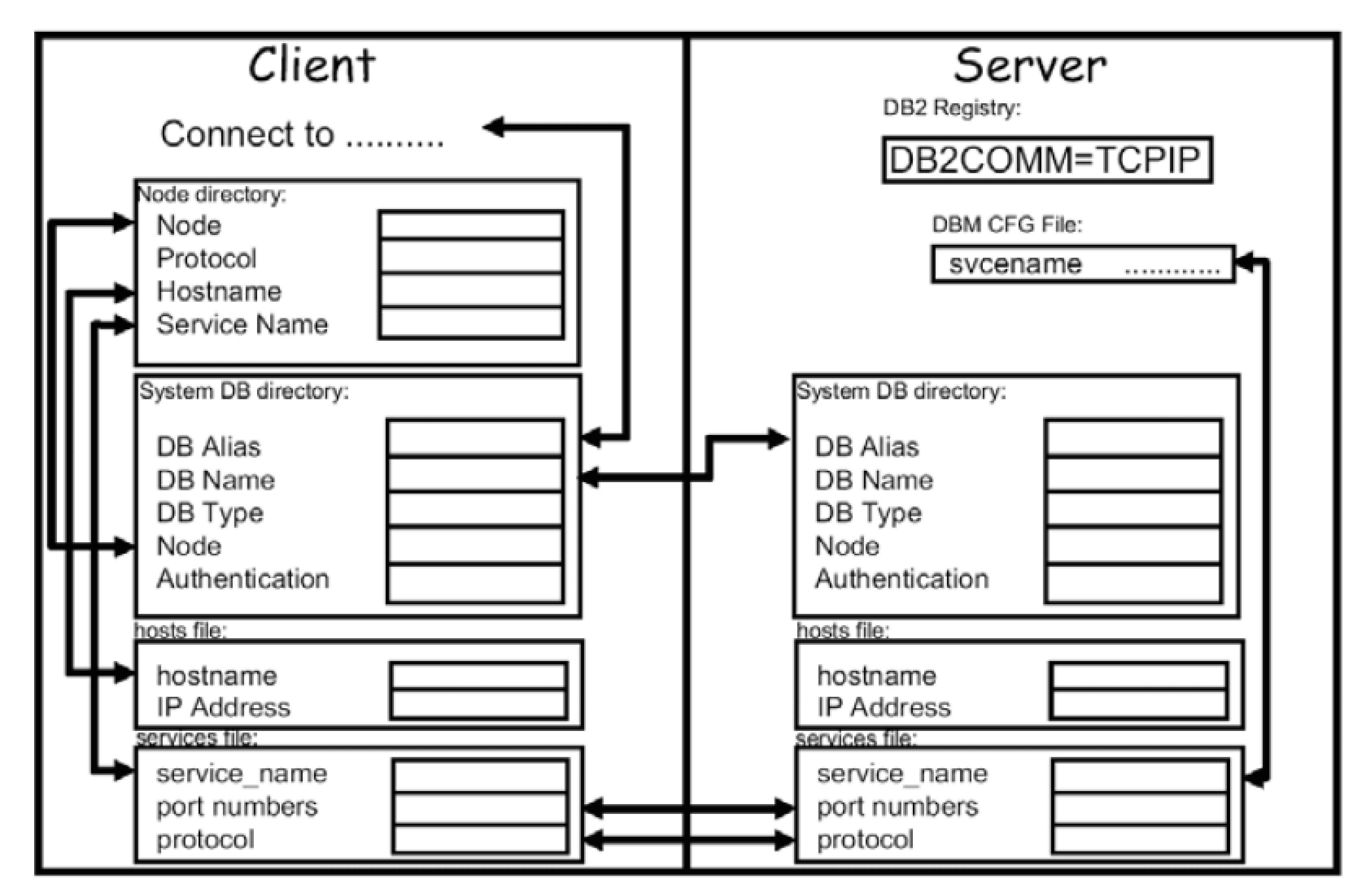


图 4-50 客户端到服务器端通信的配置参数

总的来说，要想配置客户端到服务器通信，需要经过下面几个步骤：

(1) 更新 TCP/IP 连接的 hosts 和 services 文件。

如果要建立到远程数据库服务器的连接(通过使用主机名)，但是网络没有包含 DNS(域名服务器，用来解析主机名到 IP 地址)，那么必须更新 hosts 文件。如果要通过 IP 地址访问远程数据库服务器，那么不需要此步骤。如果要在建立与远程数据库服务器的连接时指定连接服务名称，那么需要更新 services 文件。连接服务是表示连接端口号的任意名称。如果要访问远程数据库服务器的端口号，那么不需要此步骤。

要更新客户机上的 hosts 文件以将远程服务器的主机名解析为 IP 地址，可以使用文本编辑器在 hosts 文件中添加条目，作为服务器的 IP 地址。例如：

```
    11.21.15.235     myserver   # IP address for myserver
```

其中，11.21.15.235 表示 ip_address，myserver 表示 hostname。

要更新客户机上的 services 文件以将服务名称解析为远程服务器的端口号，可以使用文本编辑器将“连接服务名称”和端口号添加到 services 文件中。例如：

```
    server1  50000/tcp  #DB2connection service port
```

其中，server1 表示连接服务名称，50000 表示连接端口号(50000 为默认值)。

(2) 使用 CLP 从客户机编目 TCP/IP 节点。

编目 TCP/IP 节点会在数据在服务器的客户机节点目录中添加用于描述远程节点的条目。此条目指定客户机访问远程主机时选择的别名(node_name)、hostname(或 ip_address)和 svcename(或 port_number)。catalog 命令如下：

```
    db2 catalog TCPIP node node_name remote hostname|ip_address server
service_name|port_number [remote_instance instance_name]  [system system_name]
[ostype  os_type]   -----[]内的选项是可选的
    db2  terminate
```

要编目端口号为 50000、主机名为 server1、节点名为 db2node 的节点，应从 DB2 提示符处输入以下内容：

```
    db2  catalog TCPIP node db2node remote server1 server 50000
    DB20000I  CATALOG TCPIP NODE 命令成功完成。
    DB21056W  直到刷新目录的高速缓存之后，目录更改才会生效
    db2  terminate
    DB20000I  TERMINATE 命令成功完成
```

(3) 使用 CLP 从客户机编目数据库。

必须先在客户机上编目数据库，客户机应用程序才能访问远程数据库。创建数据库时，

除非指定不同的数据库别名，否则将自动在服务器上以与数据库名称相同的数据库别名编目数据库。在数据服务器客户机上使用数据库目录中的信息和节点目录中的信息(除非要编目不需要节点的本地数据库)来建立与远程数据库的连接。在编目远程数据库时需要数据库名称、数据库别名、节点名、认证类型(可选)、注释(可选)等信息。catalog db 命令如下：

```
    db2 catalog database database name as database alias at  node node name
[ authentication auth_value ]
```

要在使用认证 server1 的节点 *db2node* 上编目名为 *sample* 的远程数据库，以便具有本地数据库别名 *mysample*，可输入下列命令：

```
    db2  catalog database sample as mysample at node db2node  authentication server
    db2  terminate
```

(4) 用于编目数据库的参数表。

使用表 4-3 记录编目数据库所需的参数值。

表 4-3　编目数据库所需的参数

参　数	描　述	样　本　值
数据库名称(database_name)	创建数据库时，除非另有指定，请将数据库别名设置为数据库名称。例如，在服务器上创建 sample 数据库时，还将创建数据库别名 sample。数据库名称表示远程数据库别名(在服务器上)	sample
数据库别名(database_alias)	表示远程数据库的任意本地别名。如果没有提供别名，那么默认名称与数据库名称(database_name)相同。当从客户机连接至数据库时，使用此名称	mysample
认证(auth_value)	在你的环境中所需的认证类型	server
节点名(node_name)	用来描述数据库存放位置的节点目录条目的名称。对用来编目节点的节点名(node_name)使用相同的值	db2node

(5) 使用 CLP 测试客户机至服务器的连接。

在编目节点和数据库之后，应连接至数据库以测试连接。在测试连接之前：

- 数据库节点和数据库必须编目
- userid 和 password 的值对于认证它们所在的系统必须有效

要测试客户机与服务器的连接，可在客户机的命令行输入以下命令以连接至远程数据库：

```
db2 => connect to database_alias user userid
```

例如：

```
db2 => connect to sample user informix using informix
```

如果连接成功，会接收到一条消息，显示已连接数据库的名称。将给出类似图 4-51 所示的消息。

现在就可以使用数据库了。例如，要检索系统目录表中所有表名的列表，可输入以下 SQL 语句：

```
db2 SELECT tabname from syscat.tables
```

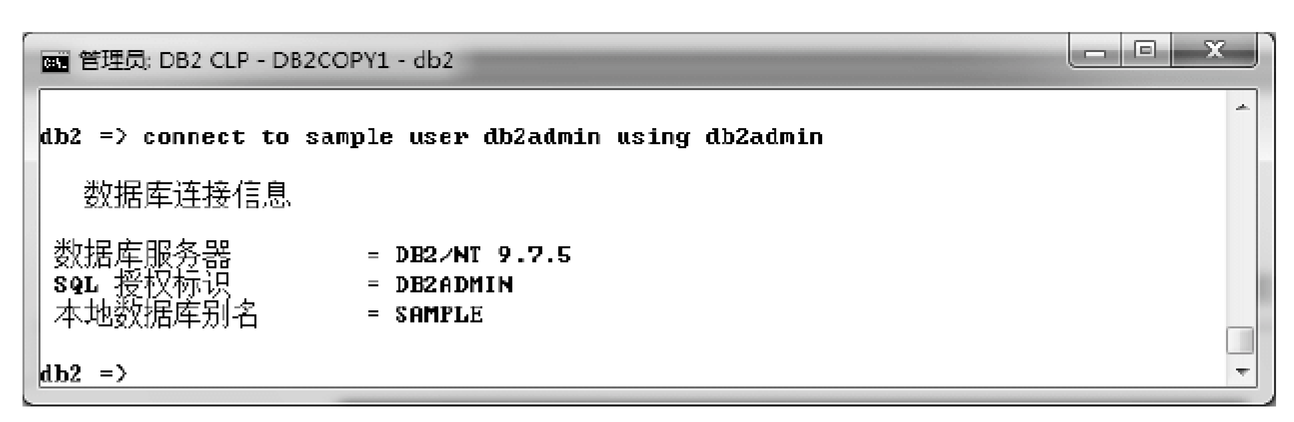


图 4-51　连接成功的消息

当结束使用数据库连接时，输入 connect reset 命令以结束数据库连接。

(6) 客户机至服务器的连接配置总结。

图 4-52 所示的这个检查列表总结了配置客户机到服务器通信时的主要检查项。

配置客户端连接服务器时的检查列表

- ☐ DB2SYSTEM注册变量已经设置为hostname，DB2安装期间默认已经设置为hostname，如果变更过hostname，请重新设置
- ☐ 在服务器上设置DB2COMM注册变量 db2set DB2COMM=tcpip
- ☐ 在services中设置实例端口，注意不要和别的端口冲突
- ☐ 更新实例的配置文件，update dbm cfg using svcename db2c_db2
- ☐ 启动实例，用netstat命令检查端口状态是否为listening状态
- ☐ 如果使用CA，需要启动DAS；如果使用CLP，在客户端catalog节点 catalog tcpip node n1 remote hostname svcename 50000
- ☐ 在客户端catalog数据库，在我们已经编目的节点上 catalog db sample at node n1
- ☐ 在客户端提供用户名和密码测试能否连接数据库；连接后如果能够读取数据，就说明配置连接成功

图 4-52　检查列表

上面演示了配置客户机到服务器 TCP 通信的过程，其实在实际生产中，除了 TCP 通信，还有 APPC、APPN 和 NETBIOS 等很多通信协议。但是这些通信协议我们都不常用到，在这里就不讲解了。而且在某些环境中，我们还会遇到一些其他组件：

- DB2 Connect 网关。这里指的是 DB2 Connect 服务器产品，该产品提供了一个网关，IBM 数据服务器客户机可通过该网关连接到中型机和大型机产品上的 DB2 服务器。
- LDAP(轻量级目录访问协议)。在启用了 LDAP 的环境中，不必配置客户机至服务器通信。当客户机试图连接至数据库时，如果本地机器的数据库目录中不存在该数据库，那么在 LDAP 目录中搜索连接数据库必需的信息。

当服务器设置为使用开发环境时(例如 IBM Data Studio)，可能会在初始 DB2 连接时遇到错误消息 SQL30081N。可能的根本原因是远程数据库服务器的防火墙阻止建立连接。在这种情况下，请验证是否正确地配置了防火墙来接受客户机的连接请求。

4.6 实际生产中连接数据库的各种方式

在第 1 章中，我们简单介绍了 DB2 不同类型的客户机，这里针对不同的用途，再详细说明一下。

IBM 数据服务器客户机和驱动程序的类型：

- IBM 数据服务器 JDBC 和 SQLJ 驱动程序。
- IBM 数据服务器 ODBC 和 CLI 驱动程序。
- IBM 数据服务器驱动程序包。
- IBM 数据服务器运行时客户机。
- IBM 数据服务器客户机。

每个 IBM 数据服务器客户机和驱动程序都提供特定类型的支持：

- 仅对于 Java 应用程序才使用 IBM 数据服务器 JDBC 和 SQLJ 驱动程序。
- 仅对于使用 ODBC 或 CLI 的应用程序才使用 IBM 数据服务器 ODBC 和 CLI 驱动程序(也称为 cli driver)。
- 对于使用 ODBC、CLI、.NET、OLE DB、PHP、Ruby、JDBC 或 SQLJ 的应用程序，使用 IBM 数据服务器驱动程序包。
- 对于使用 DB2CI 的应用程序，使用 IBM 数据服务器客户机。
- 如果需要 DB2 命令行处理器 Plus(CLPPlus)支持，使用 IBM 数据服务器驱动程序包。

- 要具有命令行处理器(CLP)支持以及用于运行和部署应用程序的基本客户机支持，请使用 IBM 数据服务器运行时客户机。
- 要具有用于数据库管理以及使用诸如 ODBC、CLI、.NET 或 JDBC 之类的应用程序编程接口(API)来开发应用程序的支持，请使用 IBM 数据服务器客户机。

IBM 数据服务器 JDBC 和 SQLJ 驱动程序

IBM 数据服务器 JDBC 和 SQLJ 驱动程序是 Java 存储过程和用户定义函数(UDF)的默认驱动程序。此驱动程序支持以 Java 编写的、使用 JDBC 访问本地或远程服务器的客户机应用程序和 applet，以及 Java 应用程序中的嵌入式静态 SQL 的 SQLJ。

比如，应用服务器上的 Java 程序通过 WebLogic 中间件连接数据库，理论上可以采用 Oracle 提供的 WebLogic 自带的 JDBC type 4 驱动，也可以采用 DB2 提供的第三方 JDBC 驱动。为了使连接数据库的 JDBC 驱动版本和数据库服务器的版本保持一致，在生产环境中部署时建议采用 DB2 数据库服务器上的驱动，换言之，需要把数据库服务器上、实例目录中 sqllib/java/下的 db2jcc.jar、db2jcc4.jar(可选)和 db2jcc_license_cu.jar 拷贝到应用服务器上 WebLogic 用户的$DOMAIN_HOME/lib 下。以后如果数据库版本升级后，也需要同时更新这几个文件。

那么如何确定 JDBC 驱动的版本呢？

在应用服务器上执行(当前目录中存放了 db2jcc.jar 文件等)：

```
在 AIX 服务器上:
$/usr/java6 64/bin/java -cp ./db2jcc.jar com.ibm.db2.jcc.DB2Jcc -version
    IBM DB2 JDBC Universal Driver Architecture 3.62.57
```

IBM 数据服务器 ODBC 和 CLI 驱动程序

IBM 数据服务器 ODBC 和 CLI 驱动程序是用于独立软件供应商(ISV)的轻量级部署解决方案。此驱动程序(也称为 CLI 驱动程序)为使用 ODBC API 或 CLI API 的应用程序提供运行时支持，而不需要安装 IBM 数据服务器客户机或 IBM 数据服务器运行时客户机。此驱动程序仅以 tar 文件的形式提供，不以可安装映像的形式提供，仅使用英文报告消息。

CLI 驱动程序比较适合 C 程序等直接连接数据库用，调用对应的 API，比如 sqlconnect 或 sqldriverconnect 等。

IBM 数据服务器驱动程序包

轻量级部署解决方案——IBM 数据服务器驱动程序包为使用 ODBC、CLI、.NET、OLE DB 、PHP、Ruby、JDBC 或 SQLJ 的应用程序提供运行时支持，而不需要安装 IBM 数据服务器运行时客户机或 IBM 数据服务器客户机。此驱动程序磁盘空间占用量较小，旨

在由独立软件供应商(ISV)重新分发以及用于大型企业的典型大规模部署方案中的应用程序分发。

IBM 数据服务器驱动程序包的功能包括：

- 用于动态创建、编辑和运行 SQL 语句和脚本的 DB2 命令行处理器 Plus。
- 支持使用 ODBC、CLI、PHP 或 Ruby 访问数据库的应用程序。
- 支持以 Java 编写的、使用 JDBC 的客户机应用程序和 applet，同时还支持 Java 嵌入式 SQL(SQLJ)。
- .NET、PHP 和 Ruby 的 IBM Informix 支持。
- 支持运行嵌入式 SQL 应用程序。不提供预编译器或绑定功能。
- 用于重建 PHP、Ruby、Python 和 Perl 驱动程序的应用程序头文件。在 IBM 数据

服务器驱动程序包中未提供 Python 和 Perl 驱动程序；但是可以使用这些头文件来下载和构建这些驱动程序。

- 支持 DB2 交互式调用级接口(db2cli)。
- 支持 DRDA 跟踪(db2drdat)。
- 在 Windows 操作系统中，IBM 数据服务器驱动程序包还支持使用.NET 或 OLEDB 来访问数据库的应用程序。此外，本驱动程序以可安装映像的形式提供，还提供了合并模块，从而可轻松将此驱动程序嵌入基于 Windows Installer 的安装中。

IBM 数据服务器运行时客户机

IBM 数据服务器运行时客户机提供了在远程数据库上运行应用程序的方法。GUI 工具未随 IBM 数据服务器运行时客户机一起提供。功能包括：

- 用于发出命令的 DB2 命令行处理器(CLP)。CLP 还提供了用于对服务器执行远程管理的基本方法。
- 用来处理数据库连接、SQL 语句、XQuery 语句和命令的基本客户机支持。
- 支持常用数据库访问接口：JDBC、ADO.NET、OLE DB、ODBC、命令行界面(CLI)、PHP 和 Ruby。此支持包括用来定义数据源的驱动程序和功能。例如，对于 ODBC，安装 IBM 数据服务器客户机会安装 ODBC 驱动程序并注册该驱动程序。应用程序开发者和其他用户可以使用“Windows ODBC 数据源管理员”工具定义数据源。
- 利用轻量级目录访问协议(LDAP)。
- 支持常用网络通信协议：TCP/IP 和“命名管道”。
- 支持在同一台计算机上安装客户机的多个副本。这些副本可以是相同的版本，也可以是不同的版本。

- 许可条款允许随应用程序自由重新分发 IBM 数据服务器运行时客户机。
- 从所需的安装映像大小和磁盘空间来看，比完整部署 IBM 数据服务器客户机的占用量更小。
- 存储用于与数据库和服务器连接的信息的目录。

IBM 数据服务器客户机

IBM 数据服务器客户机具有 IBM 数据服务器运行时客户机的所有功能，以及用于数据库管理、应用程序开发和客户机/服务器配置的功能。

包括下列功能：

- 在 Windows 操作系统中修剪 IBM 数据服务器客户机映像以减小安装映像大小的功能。
- 用来帮助对数据库进行编目和配置数据库服务器的“配置助手”。
- 用于数据库实施和数据库管理的“控制中心”和其他图形工具。
- 适用于新用户的“第一步”文档。
- Visual Studio 工具。
- 应用程序头文件。
- 各种编程语言的预编译程序。
- 绑定支持。
- 样本和教程。
- IBM Informix 支持 PHP、Ruby、.NET、JCC 和 JDBC。

4.7　案例：数据库连接问题诊断

下面以实际生产中遇到的数据库连接问题为例来说明针对该类问题的解决思路，供读者参考。

现象：

某项目组向 DBA 组报数据库故障，197.3.135.62 上的数据库通过远程方式无法连接，而 197.3.135.62 本机却可以连接。并且昨天还好好的，今天就突然不行了。

解决思路：

首先，先了解问题并重现问题。

在 197.3.135.62 上可以连接本地数据库：

```
$db2 connect to pesdb
   Database Connection Information
```

```
 Database server        = DB2/AIX64 9.5.8
 SQL authorization ID   = DB2PESJQ
 Local database alias   = PESDB
```

检查必要的参数配置，均正确无误：

```
$db2set
DB2COMM=TCPIP
$db2 get dbm cfg |grep -i svce
TCP/IP Service name                          (SVCENAME) = DB2 db2pesjq
$cat /etc/services|grep -i DB2 db2pesjq
DB2 db2pesjq    60000/tcp
DB2 db2pesjq 1  60001/tcp
DB2 db2pesjq 2  60002/tcp
DB2_db2pesjq_END       60003/tcp
```

重现远程连接数据库失败的问题：在另外一台服务器 197.3.137.200 上，catalog 197.3.135.62 上的数据库 PESDB：

```
    DS8K:/home/db2inst1$db2 catalog tcpip node PESPJQ remote 197.3.135.62
server 60000

    DB20000I  The CATALOG TCPIP NODE command completed successfully.

    DB21056W  Directory changes may not be effective until the directory cache
is refreshed.

    DS8K:/home/db2inst1$db2 list  node directory
     Node Directory
     Number of entries in the directory = 1

    Node 1 entry:

     Node name                      = PESPJQ
     Comment                       =
     Directory entry type             = LOCAL
     Protocol                      = TCPIP
     Hostname                      = 197.3.135.62
     Service name                   = 60000

    DS8K:/home/db2inst1$db2 catalog database pesdb as pesdb at node PESPJQ

    DB20000I  The CATALOG DATABASE command completed successfully.

    DB21056W  Directory changes may not be effective until the directory cache
is refreshed.
    DS8K:/home/db2inst1$db2 list db directory
```

```
 System Database Directory

 Number of entries in the directory = 1

Database 1 entry:

 Database alias                       = PESDB
 Database name                        = PESDB
 Node name                           = PESPJQ
 Database release level                 = d.00
 Comment                             =
 Directory entry type                  = Remote
 Catalog database partition number      = -1
 Alternate server hostname              =
 Alternate server port number            =
```

连接 PESDB，报错 SQL30081N，说明通信存在错误：

```
DS8K:/home/db2inst1$db2 connect to pesdb user XXXX  using XXXXX

SQL30081N  A communication error has been detected. Communication protocol

being used: "TCP/IP".  Communication API being used: "SOCKETS".  Location
where the error was detected: "197.3.135.62".  Communication function
detecting the error: "recv".  Protocol specific error code(s): "", "", "0".

SQLSTATE=08001
```

为了测试通信情况，我们直接 telnet 197.3.135.62 的 60000 端口，发现建立的连接会被瞬间关闭，说明到数据库的 6000 端口的连接存在问题：

```
DS8K:/home/db2inst1$telnet 197.3.135.62 60000

Trying...
Connected to 197.3.135.62.
Escape character is '^]'.
Connection closed.
```

然后，对问题进行分析。

通过上面的现象，基本可以初步定位为数据库服务器的 60000 端口连接有问题。

于是，在 197.3.135.62 上分析 60000 端口的占用情况：

```
#netstat -an|grep -i 60000|grep -i listen

tcp4       0       0  *.60000                 *.*                    LISTEN
```

端口 60000 处于侦听状态，那么是哪个程序占用了 60000 端口并且在侦听呢？

```
    #lsof -i:60000

    lsof: WARNING: compiled for AIX version 6.1.3.0; this is 6.1.0.0.

    COMMAND      PID     USER   FD   TYPE            DEVICE SIZE/OFF NODE NAME

    WSH      7078024   pesapp    8u  IPv4 0xf1000e00053e2bb0      0t0  TCP
PESPJQ:DB2_db2pesjq (LISTEN)

    db2sysc 44105780 db2pesjq  8u  IPv4 0xf1000e00056233b0      0t0  TCP
*:DB2_db2pesjq (LISTEN)

    db2sysc 44105780 db2pesjq   10u  IPv4 0xf1000e00057ccbb0    0t0  TCP
PESPJQ:DB2_db2pesjq->197.3.64.121:48440 (ESTABLISHED)

    db2sysc 44105780 db2pesjq   11u  IPv4 0xf1000e0002b73bb0    0t0  TCP
PESPJQ:DB2_db2pesjq->197.3.107.49:65045 (ESTABLISHED)
```

我们看到如下奇怪的现象：除了数据库进程 db2sys 在 60000 端口侦听之外，WSH 程序也占用了 60000 端口，是昨天晚上 21:05 启动的，如下所示：

```
    #ps -ef|grep 7078024

    pesapp  7078024 39845996   0 21:05:42      -  0:00 WSH -c 11 -i 0 -s 136315169
-p 2048 -P 65535
```

WSH 是 tuxedo 的进程，WSH 用的端口是随机分配的，-p 指定了最小端口，-P 指定了最大端口。针对上面的情况，可以分配 2048~65535 之间的随机端口，因为分配了 60000 端口，所以与数据库端口发生冲突。

最后，解决问题。在将 WSH 进程重启后，为之随机分配了其他端口 57982 和 57983，释放对 60000 端口的占用，数据库可以正常连接了。当然，基本解决办法是规范 tuxedo 程序占用的端口范围，避开数据库端口。

```
    pesapp:/home/pesapp/>lsof |grep -i listen|grep -i pesapp|grep -i wsh

    lsof: WARNING: compiled for AIX version 6.1.3.0; this is 6.1.0.0.
    lsof: WARNING: access /home/pesapp/.lsof PESPJQ: No such file or directory
    lsof: WARNING: can't open /home/pesapp/.lsof_PESPJQ: Permission denied

    WSH       2294340 pesapp    8u  IPv4 0xf1000e0005061bb0             0t0
TCP PESPJQ:57983 (LISTEN)

    WSH       3867244 pesapp    8u  IPv4 0xf1000e00055603b0             0t0
TCP PESPJQ:57982 (LISTEN)
```

4.8　本章小结

本章讲解了访问数据库的各种接口，重点讲解了如何配置客户端到服务器通信。一旦配置好访问数据库的接口，就可以访问数据库来创建数据库对象。接下来介绍如何创建数据库对象。



创建数据库对象

在数据库创建后，可以根据业务需求来设计和创建数据库对象。可以在 DB2 数据库中创建下列数据库对象：

- 模式
- 表
- 约束
- 索引
- 序列
- 视图
- 触发器
- 例程

可以使用图形用户界面或通过显式执行 SQL 语句来创建这些数据库对象。用于创建这些数据库对象的语句称为“数据定义语言”(DDL)，它们通常以关键字 CREATE 或 ALTER 作为前缀。

5.1 模式

5.1.1 模式的概念

数据库中的大多数对象都指定由两部分组成的唯一名称，如图 5-1 所示。第一部分(最左边的)称为限定词或模式，而第二部分(最右边的)称为简单(或未限定)名称。从句法上来说，这两部分并置成用句点分隔的单个字符串。第一次创建可以由模式名限定的任何对象(例如表、索引、视图、用户定义的数据类型、用户定义的函数、程序包或触发器)时，会根据对象名称中的限定词将对象指定给特定模式。

Schema.OjbectName

图 5-1　数据库对象名的组成

例如，图 5-2 说明在创建表的过程中如何将表指定给某个特定模式。

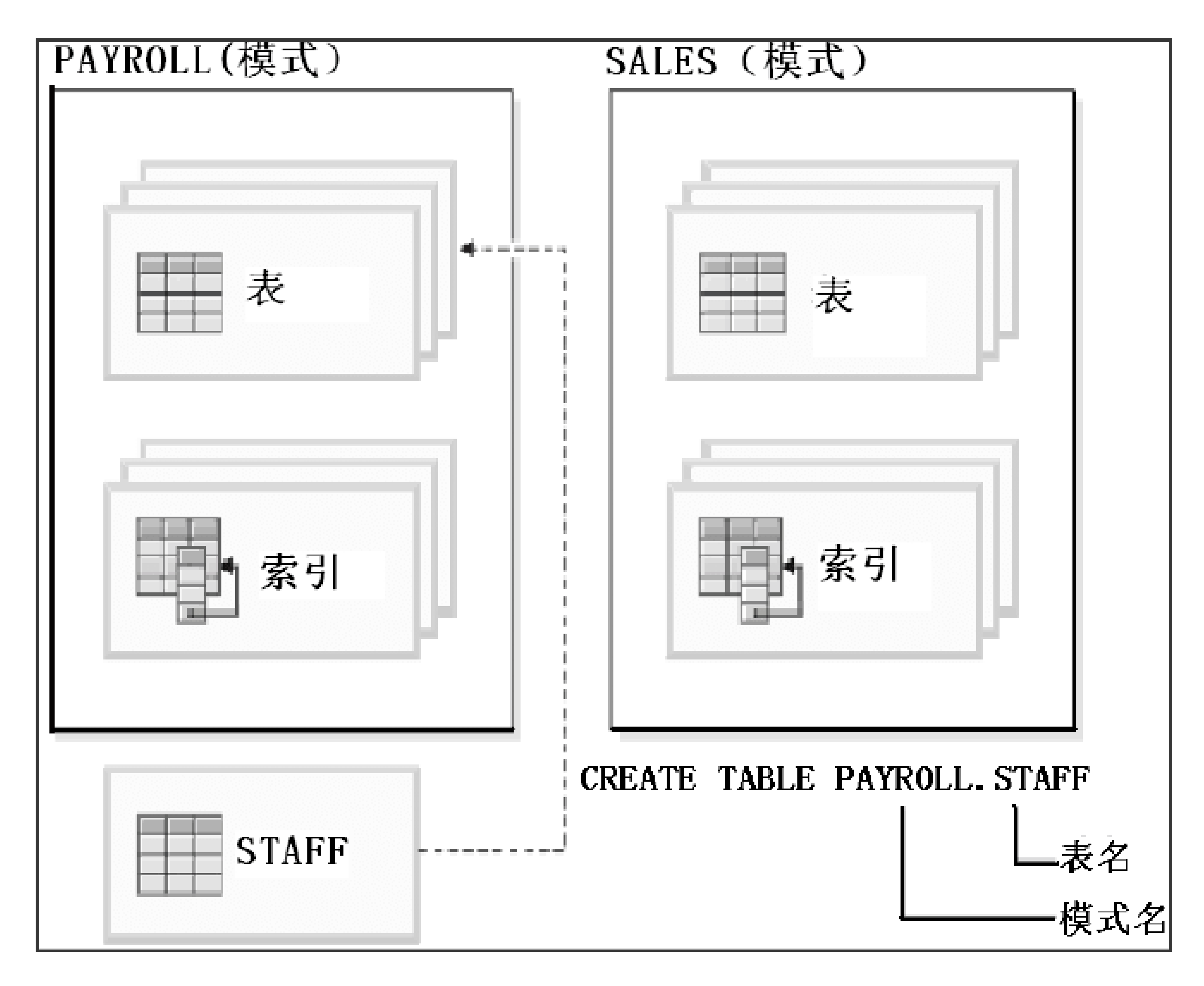


图 5-2　创建表并将其指定给某个特定模式

DB2 中的模式(schema)是已命名对象的集合，本身也是数据库对象，提供了一种方法来按逻辑分组这些对象。这些对象包括表、视图、索引、触发器、函数、程序包、例程等其他对象。模式提供了数据库中对象的逻辑类别。模式也是名称限定词；提供了一种方法来对几个对象使用相同名称，并防止对这些对象进行二义性引用。例如，使用模式名“PROD”和“DEV”很容易区分两个不同的 SALES 表(PROD.SALES 和 DEV.SALES)。模式名的最大长度为 128 字节，用作两部分对象名的第一部分。例如名称 CITIC.CUSTOMER，在这个示例中，CUSTOMER 表的完全限定名包含模式名 CITIC，这可以在系统编目中将之与其他名为 CUSTOMER 的表区分开。可以把模式想象为特定对象的创建者、生成者和主人。

如果创建对象而没有指定模式，那么对象会使用用户名与某个隐式模式相关联(假设用户或组具有 IMPLICIT_SCHEMA 数据库权限，IMPLICIT_SCHEMA 权限简单来说就是假设

用户在创建对象的时候没有使用模式，那么数据库就隐含地创建一个和用户名一样的模式，关于这个权限在 14 章中有详细讲解)。当 SQL 语句引用对象时，如果没有指定模式名，那么也会隐式地加上调用者的用户名。

5.1.2　系统模式

对于每个数据库，都创建和维护一组系统编目表。这些表包含关于数据库对象(例如表、视图、索引和包)的定义信息以及关于用户对这些对象的访问类型的安全信息。这些表存储在 SYSCATSPACE 表空间中，并采用保留的系统模式名：

- **SYSIBM、SYSFUN** 和 **SYSPROC**：一组例程，包括函数和存储过程，其中的 SYSIBM 是基本系统编目的模式(不建议直接访问)。
- **SYSCAT**：一组只读的系统编目表视图，记录数据库对象的结构信息。
- **SYSSTAT**：一组可更新的编目视图。这些可更新的视图允许更新某些统计信息，从而模拟和测试数据库的性能，或者更新统计信息而不使用 RUNSTATS 实用程序。
- **SYSIBMADM**：一组动态性能视图，可以从该组视图中获取数据库的性能运行信息。在本书“第 9 章：DB2 基本监控方法”中有关于性能视图的详细讲解和案例。

5.1.3　设置和获得当前模式

在客户端连接实例或数据库时，会话的特殊寄存器 CURRENT SCHEMA 包含默认的限定符，用于对特定 DB2 连接中发出的动态 SQL 语句所引用的未限定对象进行限定。初始值等于特殊寄存器 USER 中的值(运行时用户)。静态 SQL 语句(在默认情况下)由绑定应用程序的用户的授权 ID 进行限定。用户可以使用 SET CURRENT SCHEMA 语句修改特殊寄存器 CURRENT SCHEMA 的值。

可以使用 VALUES CURRENTSCHEMA 或 SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1 命令获得当前的模式名。下面来看两个使用模式的示例：

例 5-1　用户= HRUSER01，具有 IMPLICIT_SCHEMA 权限。

命令	结果
CREATE TABLE TEST1(ID INT, NAME VARCHAR(25))	Table HRUSER01.TEST1 created
CREATE TABLE CITIC.TEST1(ID INT, NAME VARCHAR(25))	Table CITIC.TEST1 created
SET CURRENT SCHEMA='CITIC'	CURRENT SCHEMA special register set to CITIC
INSERT INTO TEST1 VALUES(1,'John Doe')	Data inserted into table CITIC.TEST1

例 5-2 用户= HRUSER01，没有 IMPLICIT_SCHEMA 权限。

命令	结果
CREATE TABLE TEST1(ID INT, NAME VARCHAR(25))	SQL0552N "HRUSER01" does not have the privilege to perform operation "IMPLICIT CREATE SCHEMA". SQLSTATE=42502
CREATE TABLE HRUSER01.TEST1 (ID INT, NAME VARCHAR(25))	SQL0552N "HRUSER01" does not have the privilege to perform operation "IMPLICIT CREATE SCHEMA". SQLSTATE=42502
CREATE SCHEMA HRUSER01 AUTHORIZATION HRUSER01	Schema HRUSER01 created
CREATE TABLE TEST1(ID INT, NAME VARCHAR(25))	Table HRUSER01.TEST1 created

5.1.4 模式和用户的区别

你需要把模式和用户区分开来，默认情况下用户(用户拥有 IMPLICIT_SCHEMA 权限)拥有与之同名的模式，也可以根据需要创建模式并授权给某个用户。模式创建有隐式创建和显式创建两种方式。

隐式创建

如果具有 IMPLICIT_SCHEMA 权限，那么可以隐式地创建模式。只要具有此权限，无论何时使用不存在的模式名创建对象，都会隐式创建模式。只要创建对象的用户拥有 IMPLICIT_SCHEMA 权限，通常会在第一次创建模式中的数据对象时隐式创建模式。

显式创建

使用 CREATE SCHEMA 语句创建模式。有关模式的信息保存在连接的数据库的系统目录表中。

要创建模式并让另一个用户成为该模式的所有者(后一个操作是可选的)，需要具有 SYSADM 或 DBADM 权限。即使不具有这两种权限中的任何一种，也可以使用自己的授权标识来创建模式。作为 CREATE SCHEMA 语句的一部分创建的任何对象的定义者是模式所有者。模式所有者可以授予和撤销其他用户的模式特权。

要通过命令行创建模式，请输入以下语句：

```
CREATE SCHEMA <schema-name> [ AUTHORIZATION <schema-owner-name> ]
```

其中，<schema-name>是模式的名称。此名称在目录中已记录的模式内，必须唯一，并

且不能以 SYS 开头。如果指定可选的 AUTHORIZATION 子句，那么<schema-owner-name>将成为模式所有者。如果未指定此子句，那么发出此命令的授权标识将成为模式所有者。

例 5-3　下面的示例创建了 agent 模式并且把 agent 授权给 db2inst1 用户所有。

```
CREATE SCHEMA agent AUTHORIZATION db2inst1
```

删除模式

在删除模式之前，必须删除模式中的所有对象或将它们移至另一个模式。当尝试 DROP 语句时，模式名必须在目录中；否则会返回错误。

要使用命令行删除模式，请输入：

```
DROP SCHEMA <name> RESTRICT
```

在以下示例中，删除了模式“agent”：

```
DROP SCHEMA agent RESTRICT
```

RESTRICT 关键字强制执行如下规则：不能在指定的模式中为要从数据库中删除的模式定义对象，并且必须指定 RESTRICT 关键字。

5.2　表设计

表是数据库管理器维护的逻辑结构，所有数据都存储在数据库的表中。表由不同数据类型的一列或多列组成。数据存储在行(或称为记录)中。本节不会过多地讲解 CREATE TABLE、ALTER TABLE 或 DROP TABLE 之类的命令。这些命令可以查 SQL 参考手册，本节主要讲一些和表设计相关的考虑事项，因为很多时候如果在创建表的时候没有注意到这些，一旦系统上线，后期的调整往往非常麻烦，所以在创建表之前，要做好规划设计。

5.2.1　选择合适的数据类型

定义列时，需要对列进行命名，定义这些列中将包含的数据的类型(称为数据类型)，并定义要创建的表中每列的数据长度。DB2 提供了一套丰富且灵活的数据类型。DB2 附带 INTEGER、CHAR、DATE 和大对象等基本数据类型，还提供了创建用户定义的数据类型(UDT)的工具，使用户能够创建复杂的非传统的数据类型，从而适应当今复杂的编程环境。在给定的情况下，选用哪种数据类型取决于列中存储的信息的类型和范围。

内置的数据类型分为 5 类：数字、字符串、大对象、日期时间和 XML。

用户定义的数据类型分为：单值类型、结构化类型和引用类型(一般不用)。

DB2 内置的数据类型如图 5-3 所示。

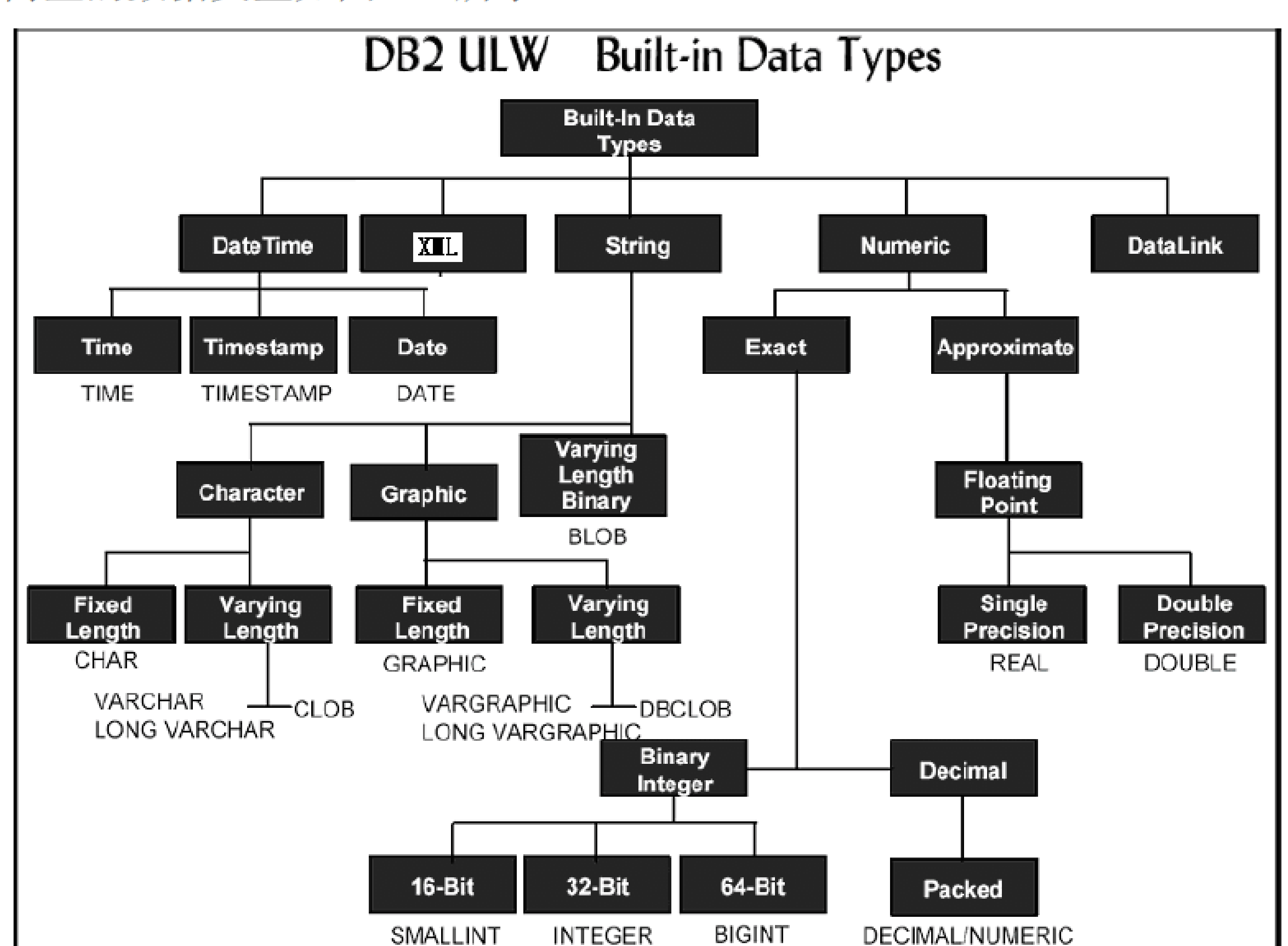


图 5-3　DB2 内置的数据类型

其中，XML 数据类型是 DB2 V9 以后版本提供的数据类型。DB2 提供了 XML 数据类型来存储格式良好的 XML 文档。XML 数据类型用于定义表中存储 XML 值的列，这些列中存储的所有 XML 值必须是结构良好的 XML 文档。引入 XML 数据类型能够将结构良好的 XML 文档以本机分层格式存储在数据库中其他关系数据旁边。

XML 列中的值存储为与字符串数据类型不同的内部表示。要在 XML 数据类型的列中存储 XML 数据，需要使用 XMLPARSE 函数对数据进行转换。可以使用 XMLSERIALIZE 函数将 XML 数据类型的值转换为 XML 文档的串行化字符串值。DB2 还提供了许多其他的内置函数来操纵 XML 数据类型。

在创建表时为列选择数据类型时一定要注意下面几点：

- 要根据业务需求选择合适的数据类型，避免出现数据类型转换。例如，笔者曾经看到有的客户使用字符来存放日期、时间戳，最后还要在程序中使用日期转换函数 to_date 进行数据类型转换，这会对应用程序带来性能影响。
- 根据需求选择合适的长度。例如，使用字段 empno 存储员工号，SMALL INT 就可以满足要求，但是如果使用 INT，就会造成两个字节的浪费。

- 如果某个字段的内容都是数字，建议大家选用整数而不要选用 CHAR。一个占用 4 字节的 INT 类型字段就可以表达到 2147483647，如果使用 CHAR 类型，那么至少需要 10 个字节。一个占用 8 字节的 BIGINT 类型字段就可以表达到 9223372036854775807，如果使用 CHAR 类型，那么至少需要 19 个字节。
- CHAR 和 VARCHAR 的选择，如果一列数据有变化，但是变化不大时，而我们又追求性能，建议使用 CHAR 类型，因为 VARCHAR 的读取性能要分两个步骤，先读长度再读数据，这比 CHAR 的性能要弱些。
- LONG VARCHAR、BLOB、CLOB 和 CBLOB 数据类型的选择，这些大对象数据类型的读取是不经过内存而直接读取的，所以可根据情况看是否能够用 VARCHAR 字段代替。
- 如果使用大对象数据类型，考虑是否对大对象列记录日志 NOT LOGGED。
- 考虑把大对象数据列单独存放在独立的表空间中，将索引数据分隔存放。

下面看看列在磁盘上是如何布局的。如果创建只有定长列的表，将严格按照 CREATE 语句中指定的顺序安排它们，如图 5-4 所示。

```
CREATE TABLE TESTORD(COL1 INT, COL2 CHAR(5), COL3 DEC(10,2), COL4 FLOAT)
```

col1 int	col2 char(5)	col3 dec(10,2)	col4 float

图 5-4　定长列的表在磁盘上的布局

如果表拥有变长列(如 VARCHAR)，那么列仍然按照 CREATE TABLE 语句中指定的顺序排序，但可变数据本身在行的末尾，如图 5-5 所示。

```
CREATE TABLE TESTORD(COL1 INT, COL2 VARCHAR(5), COL3 DEC(10,2))
```

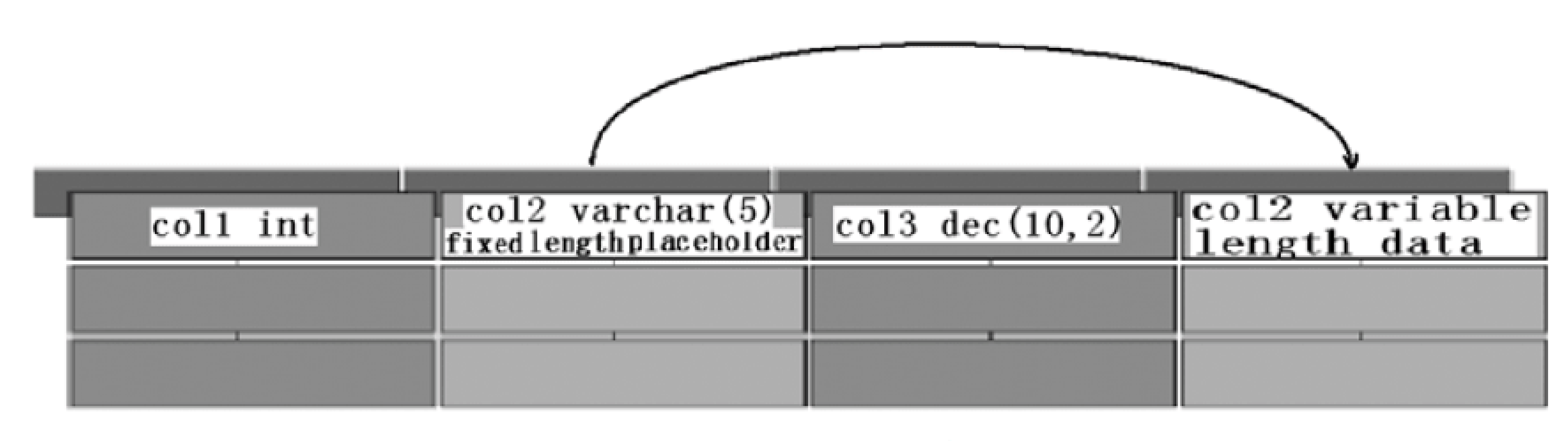


图 5-5　拥有变长列的表在磁盘上的布局

如果表中拥有长字段，那么长字段将不随每行直接插入。因为行的长度受页大小限制(4KB 到 32KB)，所以行只有一个指向长字段的指针，而将长字段与行分开放置在数据库

页中，如图 5-6 所示。

```
CREATE TABLE TESTORD(COL1 INT, COL2 CLOB(100 K), COL3 DEC(10,2))
```

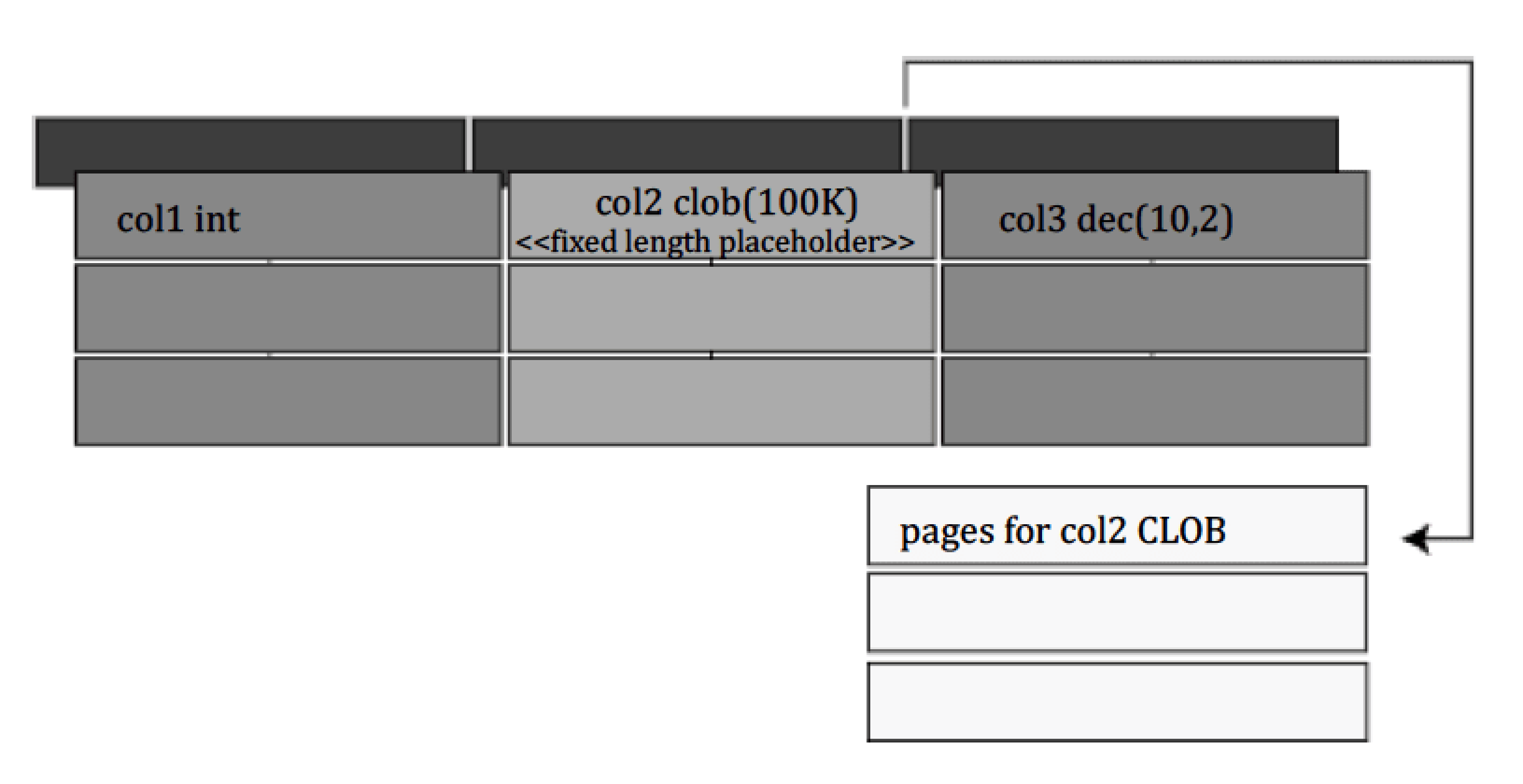


图 5-6　拥有长字段的表在磁盘上的布局

5.2.2　选择合适的约束类型

在任何业务中，数据通常必须符合特定限制或业务规则。例如，职员编号、银行支票号必须是唯一的。数据库管理器提供了约束作为强制实施这种规则的方法。约束是用于业务需求的规则。DB2 提供了下列 5 种类型的约束：

NOT NULL 约束

NOT NULL 约束防止在列中输入空值。NOT NULL 约束是这样一种规则，用于防止在表的一列或多列中输入空值。数据库中使用空值来表示未知状态。默认情况下，随数据库管理器一起提供的所有内置数据类型都支持空值的存在。但是，一些业务规则可能要求必须始终提供值(例如，乘飞机时必须提供紧急联系人信息)。NOT NULL 约束用于确保决不会为给定表列指定空值。为特定列定义 NOT NULL 约束后，尝试在该列中放入空值的任何插入或更新操作将失败。

唯一约束

唯一约束确保一组列中的值对于表中的所有行都是唯一的，且不为空。在唯一约束中指定的列必须定义为 NOT NULL。唯一约束(也称为唯一键约束)是这样一种规则，用于禁止表的一列或多列中出现重复值。唯一键和主键是受支持的唯一约束。例如，可对供应商表中的供应商标识定义唯一约束以确保不会对两个供应商指定同一供应商标识。唯一约束

确保一组列中的值对于表中的所有行都是唯一的，且不为空。在唯一约束中指定的列必须定义为 NOT NULL。数据库管理器使用唯一索引在对唯一约束的各列进行更改时强制键的唯一性。例如，DEPARTMENT 表中的典型唯一约束可以是：部门号是唯一的，且不为空。

图 5-7 显示了当表存在唯一约束时，阻止将重复的记录添加到表中。

数据库管理器在插入和更新操作期间强制执行此约束以确保数据完整性。表可以有任意数目的唯一约束，但最多将一个唯一约束定义为主键。对于同一组列，表不能有多个唯一约束。

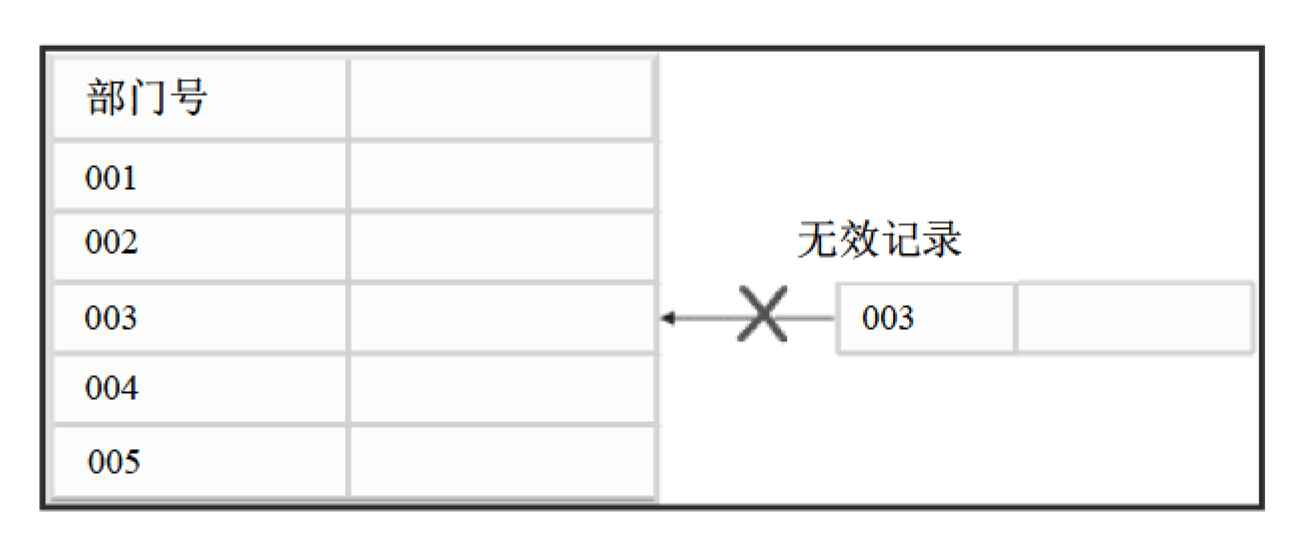


图 5-7　唯一约束能防止出现重复数据

- 当在 CREATE TABLE 语句中定义唯一约束时，唯一索引是由数据库管理器自动创建的，且被指定为主索引或系统所需的唯一索引。
- 当在 ALTER TABLE 语句中定义唯一约束且同一组列存在索引时，该索引被指定为唯一的且是系统所需的。如果这样的索引不存在，数据库管理器就会自动创建唯一索引，并将其指定为主索引或系统所需的唯一索引。

注意：

定义唯一约束与创建唯一索引是有区别的。尽管都强制唯一性，但唯一索引允许可空列，且通常不能用作参考约束的父键。

主键约束

主键是与唯一约束具有相同属性的一个列或列的组合。因为主键用来标识表中的一行，所以必须是唯一的，并且必须具有 NOT NULL 属性。一个表不能有多个主键，但可以有多个唯一键。主键是可选的，可以在创建或改变表时定义。当导出或重组数据时，主键可以对数据进行排序，所以它们也是有益的。

(表)检查约束

检查约束(也称为表检查约束)是这样一种数据库规则，用于指定表中每行的一列或多

列中允许使用的值。指定检查约束是通过限制格式的搜索条件完成的。检查约束对添加至特定表的数据设置限制。例如，表检查约束可确保每当在包含个人信息的表中添加或更新薪水数据时，职员的薪水级别至少为￥2000。

外键(参考)约束

外键约束(也称为参考约束或参考完整性约束)使你能够定义表间以及表内必需的关系。外键约束是关于一个或多个表中的一列或多列中的值的一种逻辑规则。例如，一组表共享关于公司的供应商信息。供应商的名称有时可能会更改。可定义参考约束，声明表中的供应商标识必须与供应商信息中的供应商标识相匹配。此约束会阻止外键约束，使得能够定义表间以及表内必需的关系。

例如，典型的外键约束可能规定 EMPLOYEE 表中的每个职员必须是现有部门的成员，该部门在 DEPARTMENT 表中定义。

参考完整性是数据库的一种状态，在该状态中，外键的所有值都有效。外键是表中的一列或一组列，键值需要与父表中行的至少一个主键或唯一键值相匹配。参考约束是这样一种规则，仅当满足下列其中一个条件时，外键的值才有效：

- 它们作为父键的值出现
- 为空

要建立此关系，应将 EMPLOYEE 表中的部门号定义成外键，并将 DEPARTMENT 表中的部门号定义成主键。图 5-8 显示了当两个表之间存在外键约束时，如何阻止将具有无效键的记录添加至表中。

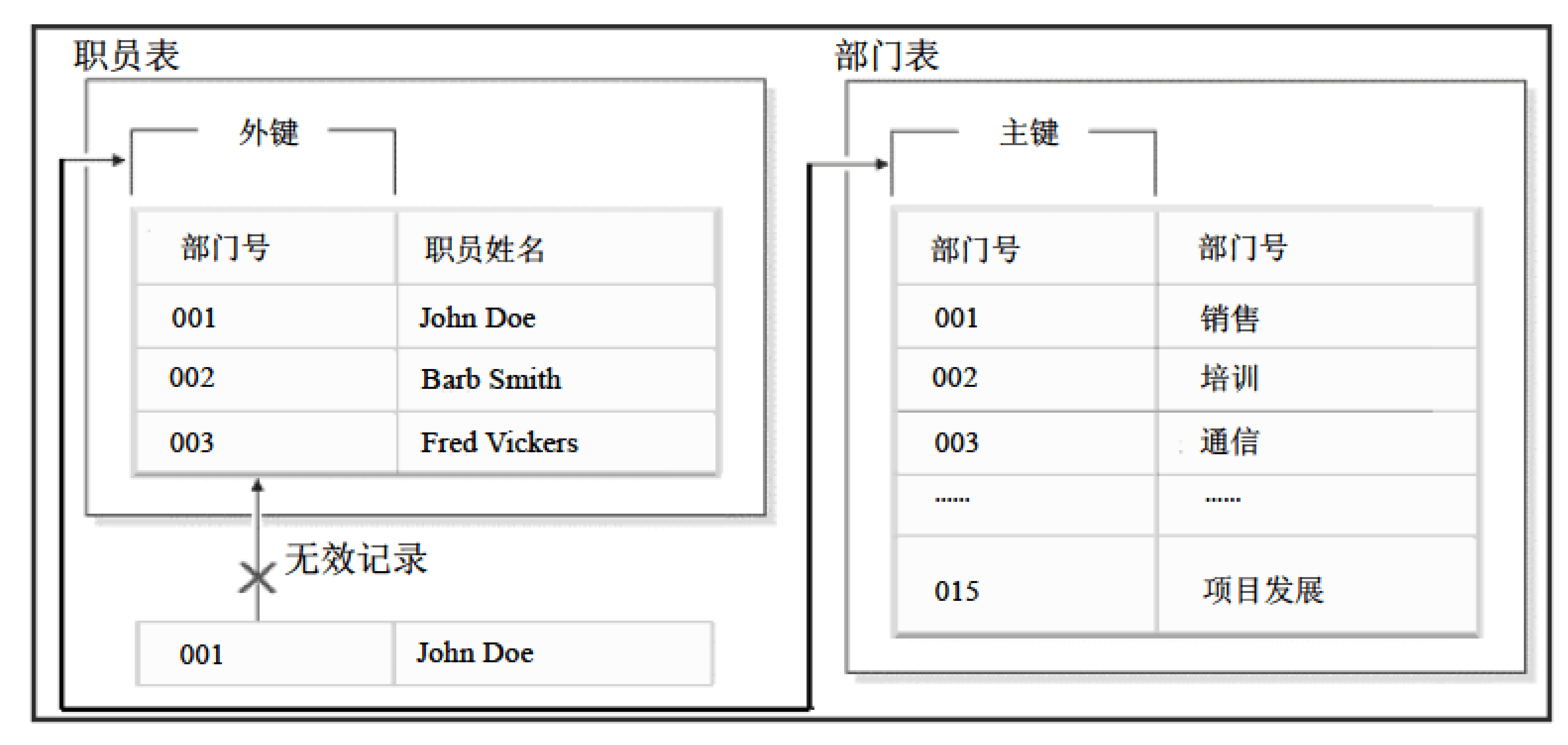


图 5-8　外键和主键约束

包含父键的表称为参考约束的父表，包含外键的表被认为是该表的从属表。

在了解了以上 5 种类型的约束后，我们在进行表的设计时，可以根据自己的业务需要

来决定创建什么类型的约束。

5.2.3　使用 not null with default

在数据库中，NULL 值是最难处理的，我们编程的时候对 NULL 值的处理也颇费周折，例如在嵌入 SQL 编程中，为了处理 NULL 值，必须在程序中声明指示符变量，这增加了编程的成本。所以建议创建表时对列使用 not null with default 选项，这样如果该列为 NULL，就使用默认值代替 NULL。

例 5-4　使用 not null with default 创建表。

```
$db2 create table xinzhuang_tab(id int NOT NULL with default 1)
```

5.2.4　生成列及应用案例

生成列在表中定义，在这些列中，存储的值是使用表达式计算得出的，而不是通过插入或更新操作指定。

当创建已知始终要使用特定表达式或谓词的表时，可对该表添加一个或多个生成列。通过使用生成列，就有机会在查询表数据时改进性能。

例如，当性能很重要时，以下两种表达式求值方式成本很高：

- 必须在查询期间进行许多次表达式求值
- 计算很复杂

为了改进查询的性能，可定义其他列，在其中将包含表达式的结果。然后，当发出包括同一表达式的查询时，可直接使用生成列；或者，优化器的查询重写组件可用生成列替换该表达式。

当查询涉及连接两个或多个表中的数据时，添加生成列允许优化器选择可能更好的连接策略。将使用生成列来改进查询的性能。结果是，可能在创建和填充表之后添加生成列。

例 5-5　创建生成列的表。在 CREATE TABLE 语句中定义生成列：

```
CREATE TABLE t1(c1 INT,
                c2 DOUBLE,
                c3 DOUBLE GENERATED ALWAYS AS(c1 + c2),
                c4 GENERATED ALWAYS AS
                  (CASE WHEN c1 > c2 THEN 1 ELSE NULL END))
```

在创建此表之后，可以使用生成列来创建索引。例如：

```
CREATE INDEX i1 ON t1(c4)
```

查询可以利用生成列。例如：

```
SELECT COUNT(*) FROM t1 WHERE c1 > c2
```

可以编写为：

```
SELECT COUNT(*) FROM t1 WHERE c4 IS NOT NULL
```

另一个示例：

```
SELECT c1 + c2 FROM t1 WHERE(c1 + c2) * c1 > 100
```

可以编写为：

```
SELECT c3 FROM t1 WHERE c3 * c1 > 100
```

对列存在的任何<column options>进一步定义该列的属性。选项包括用于防止列包含空值的 NOT NULL，用于 LOB 数据类型的特定选项，引用类型列的 SCOPE，对表的任何约束以及列的任何默认值。

5.2.5 自动编号和标识列应用案例

标识列为 DB2 提供一种方法，可自动为添加至表中的每一行生成唯一数值。当创建一个表时，如果需要将唯一标识添加至该表的每一行，那么可向该表添加标识列。要保证为添加至表的每一行提供唯一数值，应在标识列定义唯一索引，或将其声明为主键。

其他地方使用的标识列有订单号、职员编号、股票代码或事故编号。标识列的值可以"始终"或"在默认情况下"由 DB2 数据库管理器生成。

将对定义为 GENERATED ALWAYS 的标识列给予始终由 DB2 数据库管理器生成的值，不允许显式地为应用程序提供值。定义成 GENERATED BY DEFAULT 的标识列使应用程序能够显式地为标识列提供值。如果应用程序不提供值，那么 DB2 将生成一个值。因为由应用程序控制该值，所以 DB2 不能保证该值的唯一性。下面是一个 GENERATED BY DEFAULT 的示例：

```
create table db2admin.actor(
    actor id int generated by default as identity , ----用户可以输入指定值
    actor name varchar(20) ,
    act_yr_of_birth smallint)
```

要对新表定义标识列，在 CREATE TABLE 语句中使用 AS IDENTITY 子句。

例 5-6 创建标识列示例。在 CREATE TABLE 语句中定义标识列：

```
create table idn1(id integer, name char(20), dn integer NOT NULL
generated always as identity(start with 1, increment by 1)) ----DB2 自动生成值，用户无法输入指定值
```

```
对于上面的表，select * from idn1 的输出为：
ID          NAME                 DN
----------- -------------------- -----------
        1   Test                 1
        2   Test                 2
        3   Test                 3
3 条记录已选择。
```

在例 5-6 中，第 3 个列是标识列。还可以指定该列中用来在添加行时唯一标识每一行的值：在输入的第一行的列中具有值“1”；添加到该表中的每个后续行都具有相关联的值，这些值将依次增加 1。

在上面的例 5-6 中，自动生成列的当前序列值为 4。

可以使用下列命令控制生成列的当前序列值：

- 执行 alter table idn1 alter column dn restart 后，当前序列值重置为 1。
- 执行 alter table idn1 alter column dn restart with 10 后，当前序列值置为 10。

5.2.6　使用 not logged initially 特性

如果需要经常对表进行批量插入、更新和删除操作，可以考虑在创建表的时候使用 not logged initially 特性。在实际生活中，这样做对于一些临时表、stage 表非常好，可以提高批量插入、更新和删除的性能。否则，如果表中数据量很大，那么进行批量删除、插入和更新数据时会报 SQL0964C 错误，而且也比较慢。

例 5-7　使用 not logged initially 创建表。

```
db2 => create table nolog tab(id int, name char(20)) not logged initially
DB20000I SQL 命令成功完成。
db2 => delete from nolog tab ---------表中有 3 千万记录
SQL0964C 数据库的事务日志已满。 SQLSTATE=57011
```

不管在创建表时是否设定了 not logged initially 属性，都可以在交易中使用 alter table ... activate not logged initially 指定不记录日志。也可以考虑使用 activate not logged initially 清空表而不产生日志，案例如下：

```
db2 => alter  table nolog tab activate not logged initially with empty table
DB20000I SQL 命令成功完成。
db2 => commit
DB20000I SQL 命令成功完成。
```

当激活这个特性后，和 NOT LOGGED INITIALLY 语句在同一事务中的任何 INSERT、DELETE、UPDATE、CREATEINDEX、DROP INDEX 和 ALTER TABLE 产生的变化都不

会记录日志。当前事务完成后，NOT LOGGED INITIALLY 属性就会被关闭，在随后的事务中，对此表执行的所有操作都会再次被记录日志。这样不但提高了速度，也减少了日志的生成，并且减少了锁资源的使用。

我们在使用这个特性时要注意，对于一些非常重要的表，以及需要写日志的表而言，不建议使用该特性。

特别注意

NOT LOGGED INITIALLY 被激活后，如果此表相关的语句失败了(导致 rollback)，或者 ROLLBACK TO SAVEPOINT 被执行，那么整个事务将被回滚，并且被激活 NOT LOGGED INITIALLY 属性的表将被标记为不可访问，此表只能被删除并重建。所以请慎重使用此特性，并且使用此特性的时候尽可能减少出错的几率。

```
db2 "create table nolog tab(id int, name char(20)) not logged initially"
DB20000I  The SQL command completed successfully.
db2 +c "alter table nolog tab activate not logged initially"
DB20000I  The SQL command completed successfully.
db2 +c "insert into nolog tab values (1,'2')"
DB20000I  The SQL command completed successfully.
db2 +c "insert into nolog tab values (1,'2')"
DB20000I  The SQL command completed successfully.
db2 +c "insert into nolog tab values (1,'2')"
DB20000I  The SQL command completed successfully.
db2 rollback
DB20000I  The SQL command completed successfully.

db2 "select * from nolog_tab"
ID         NAME
----------- --------------------
SQL1477N  For table "DB2TEST1.NOLOG_TAB" an object "9" in table space "3"
cannot be accessed.  SQLSTATE=55019
```

5.2.7 使用 append on 特性

在数据库中，当表中数据被删除时，空间并不会释放，而是在该行原来的位置做个“DELETED”标志，表示该空间可以被重用。当 DB2 执行 INSERT 操作时，会扫描整个表的空闲空间并将新行置入空槽。而如果启用 append on 特性，那么当插入新行时，DB2 就不必搜索空槽再插入，而是直接插入到表的最后。例如：

```
CREATE TABLE appen_on_tab LIKE RECEIPTS IN SLOW_DISK_TBSP
```

可以通过将该表改变成 APPEND ON 来通知 DB2 在执行 INSERT 时不必搜索空槽：

```
ALTER TABLE appen_on_tab APPEND ON
```

这将使 INSERT 更快。这适合用于大批量追加插入一些历史表。如果启用这种特性，考虑定期 reorg 表。

注意：
当打开 append on 属性时，表不能有集群索引，否则报 SQL1581N 的错误。

5.2.8　数据、索引和大对象分开存放

在创建表的时候，考虑把表数据、索引和大对象数据分开存放到不同的表空间以提高性能。

例 5-8　创建表。

```
CREATE TABLE BOOKS( BOOKID INTEGER,
                  BOOKNAME VARCHAR(100),
                  ISBN CHAR(10) )
                  IN DATA_SPACE INDEX IN INDEX_SPACE LONG IN LONG_SPACE
```

IN、INDEX IN 和 LONG IN 子句指定将在其中存储常规表数据、索引和大对象数据的表空间。注意，这只适用于 DMS 表空间。

5.2.9　设置 pctfree

创建表时可以在每页上预留 pctfree%的可用空间，以应付未来的 row overflow。如果没有指定，默认不会预留空闲空间。通常，如果表中有很多 varchar 字段，当 varchar 字段更新时，如果更新的值比原来的长度长，并且在原来的行的 slot 无法存放该行数据时，就会在该位置留下一个指针，然后把该行插入一个新页。这样读取该行时，就会造成额外的 I/O，从而影响性能。为了解决这个问题，除了定期做碎片整理外，还可以考虑在创建表时预留一部分空间。下面是使用示例：

```
CREATE TABLE RECEIPTS
    (RECEIPT_DATE DATE NOT NULL, CUST_NUM INT NOT NULL,
     RECEIPT_KEY TIMESTAMP NOT NULL, AMOUNT DEC(10,2),
     PRIMARY KEY(CUST_NUM, RECEIPT_KEY))
     PARTITIONING KEY(CUST_NUM, RECEIPT_KEY)
```

可以使用 ALTER TABLE 语句来调整：

```
ALTER TABLE RECEIPTS PCTFREE 10     或
ALTER TABLE RECEIPTS PCTFREE 0    -----只读表
```

注意：

假如字段 name 的数据类型为 varchar(60)，如果一开始 name 长度为 10 字节，这时假设刚好可以放到数据页中。但是假设 update 操作将 name 从 10 字节更新为 60 字节，如果数据页无法放下，那么数据库就在当前位置存放指针，把数据放到新的数据页中，这就叫 overflow。overflow 会增加 I/O 读取，对性能不好。

注意：

在每个数据页面插入的第一行不受 PCTFREE 的限制。

5.2.10 表的 locksize

表的 locksize 特性与锁和并发有关，在此仅提醒大家在设计表的时候有这个特性，如果设计不当会严重影响应用程序并发。关于这个参数的详细解释需要结合锁和并发来讲解。我们会在“第 9 章：锁和并发”中详细讲解表的这个特性。

5.2.11 表的 volatile 特性

一些表具有下列特征：表的数据变化非常大，常常从空到非常大，清空后又变非常大。例如，炒股时常常需要交易委托单，这个交易委托单存放到一张表中。这张表在晚上做完批处理后清空为 0，第二天这个表又变得非常大。然后又清空。日常生活中有许多诸如此类的表。对于具有这样特征的表，请启用表的 volatile 特性。这种表被称为“易失表”。比如在金融、银行领域需要在月末进行处理的汇总表，在不长的时间范围内数据量变化特别大，从而使 RUNSTATS 得到的统计信息不准确，原因是这些统计信息只是某个时间点的信息。可以用下面这条语句把表修改为 volatile：

```
alter table transaction_log volatile cardinality ---设置银行交易流水表 volatile
```

对于这样的表，优化器在下列情况下将执行索引扫描代替表扫描，而不考虑统计信息：

- 引用的所有列都是索引的组成部分
- 索引可以在索引扫描期间应用谓词

对于类型表而言，只有类型表层次结构的根表才支持 ALTER TABLE...VOLATILE 语句。

已被标记为 volatile 的表(在 SYSCAT.TABLES 目录视图中设置了 volatile 字段的表)将不会执行自动收集统计信息(同步或异步)操作。

5.2.12　表维护相关命令

修改表

使用 ALTER TABLE 语句更改列属性，例如可空性、LOB 选项、作用域、约束、压缩属性以及数据类型等等。

要更改表，必须对要更改的表至少具有下列其中一种特权：

- ALTER 特权
- CONTROL 特权
- DBADM 权限
- 对表模式的 ALTERIN 特权

例如，在命令行中输入：

```
ALTER TABLE EMPLOYEE  ALTER COLUMN WORKDEPT  SET DEFAULT '123'
```

可以使用 ALTER TABLE 语句执行如下操作：

- 使用 DROP COLUMN 子句删除列
- 使用 ADD COLUMN 子句增加列
- 使用 ALTER COLUMN SET DATA TYPE 子句修改列属性
- 使用 SET NOT NULL 或 DROP NOT NULL 子句修改列的可空属性
- 使用 RENAME COLUMN 子句将基本表中的现有列重命名为新名称

在使用 SQL 修改这些表属性时，不再需要删除表并重新创建。这原来是很耗费时间的过程，而且在存在对象依赖时可能会很复杂。除了上述新增加的特性外，还可以使用 DB2 V9 版本以前的表修改语句：

- 增加列。增加的新列是表中的最后一列；也就是说，如果最初有 n 列，那么添加的列将是第 n+1 列。添加新列不能使所有列的总字节数超过最大记录大小。
- 修改与列关联的默认值。在定义了新默认值后，将对任何后续 SQL 操作中指示使用此默认值的列使用新值。新值必须遵守赋值规则，并且受到的限制与 CREATE TABLE 语句中记录的限制相同。

下面举几个使用 ALTER TABLE 语句修改表的例子。

例 5-9　将 Managing_Bank 列添加到 ACCOUNT 表中：

```
ALTER TABLE V9R0M0.ACCOUNT ADD COLUMN Managing_Bank VARCHAR(15)
```

例 5-10　删除 TRANSACTION 表中的 Instruction_ID 列：

```
ALTER TABLE V9R0M0.TRANSACTION DROP COLUMN Instruction_ID
```

例 5-11 将 ACCOUNT 和 TRANSACTION 表中 Account_ID 列的数据类型从 SMALLINT 改为 INTEGER：

```
ALTER TABLE dev.ACCOUNT ALTER COLUMN Account ID SET DATA TYPE INTEGER
ALTER TABLE dev.TRANSACTION ALTER COLUMN Account_ID SET DATA TYPE INTEGER
```

例 5-12 删除 ACCOUNT 表中 Credit_Line 列的 NOT NULL 属性：

```
ALTER TABLE dev.ACCOUNT ALTER COLUMN Credit_Line DROP NOT NULL
```

例 5-13 增加 TRANSACTION 表中 Description 列的大小：

```
ALTER TABLE dev.TRANSACTION ALTER COLUMN Description SET DATA TYPE VARCHAR(60)
```

例 5-14 修改 t1 表中 colnam1 列的默认值：

```
ALTER TABLE t1 ALTER COLUMN colnam1  SET DEFAULT '123'
```

重命名表

可以使用 RENAME 语句重命名现有表。例如：

```
$db2 rename table tta to rn tab
DB20000I  SQL 命令成功完成。
```

重命名表时，源表不能在任何现有定义(视图或具体化查询表)、触发器、SQL 函数或约束中引用。也不能具有任何生成列(标识列除外)，不能是父表或从属表。目录条目将更新以反映新表名。

注意：

目标表必须使用源表的模式名，不能为目标表指定其他模式名。

查看表信息

可以使用表 5-1 所示的命令获取表信息。

表 5-1 用来获取表信息的命令

命　　令	描　　述
list tables	列出用于当前用户的表
list tables for all	列出数据库中定义的所有表
list tables for schema schemaname	列出指定模式中的表

(续表)

命　　令	描　　述
list tables for schema	列出以当前用户名为模式的表
describe table tablename	显示指定的表的结构

例如，下面的命令：

```
db2 describe table department
```

产生图 5-9 所示的输出。

Column name	Type schema	Type name	Length	Scale	Nulls
DEPTNO	SYSIBM	CHARACTER	3	0	No
DEPTNAME	SYSIBM	VARCHAR	29	0	No
MGRNO	SYSIBM	CHARACTER	6	0	Yes
ADMRDEPT	SYSIBM	CHARACTER	3	0	No
LOCATION	SYSIBM	CHARACTER	16	0	Yes

图 5-9　describe table degartment 命令的输出信息

删除表

可以使用 DROP TABLE 语句删除表。当删除表时，也会删除 SYSCAT.TABLES 系统目录中包含有关表信息的那一行，并会影响从属于表的任何其他对象。例如：

- 会删除所有的列名。
- 会删除基于该表的任何列创建的索引。
- 将基于该表的所有视图标记为不可用。
- 删除的表和从属视图的所有特权被隐式撤销。
- 会删除在其中为父表或从属表的所有引用约束。
- 从属于删除的表的所有程序包和高速缓存的动态 SQL 和 XQuery 语句被标记为无效，并且这一状态会保持至重新创建了从属对象为止。这包括这样的一些程序包，它们从属于将被删除的层次结构中子表的任何超表。
- 将从属于要删除表的所有触发器标记为不可用。

要使用命令行删除表，请输入：

```
DROP TABLE <table_name>
```

以下语句可删除 DEPARTMENT 表：

```
DROP TABLE DEPARTMENT
```

CREATE TABLE …LIKE

如果想创建和原来表结构一样的表，可以使用 CREATE TABLE …LIKE 命令。例如，创建和表 employee 结构一样的表：

```
CREATE TABLE emp LIKE employee
```

注意：

如果源表有任何的唯一约束、外键约束、触发器或索引，那么由 CREATE TABLE …LIKE 产生的新表不会自动创建这些附属对象。

获取表的 DDL 信息

可以在控制中心，右击要导出 DDL 信息的表的名称，单击“生成 DDL”以导出创建表的 DDL 信息，如图 5-10 所示。



图 5-10　生成表的 DDL 信息

也可通过 db2look 命令获取创建表的 DDL 信息，例如：

```
$db2look -d sample -e -t emp2
-- No userid was specified, db2look tries to use Environment variable USER
-- USER is: T1
-- The db2look utility will consider only the specified tables
-- Creating DDL for table(s)
-- This CLP file was created using DB2LOOK Version "9.7"
-- Timestamp: Thu Aug 23 10:32:55 BEIST 2012
-- Database Name: SAMPLE
-- Database Manager Version: DB2/AIX64 Version 9.7.6
-- Database Codepage: 1208
-- Database Collating Sequence is: IDENTITY

-- Binding package automatically ...
```

```
-- Bind is successful
-- Binding package automatically ...
-- Bind is successful

CONNECT TO SAMPLE;

------------------------------------------------
-- DDL Statements for table "T1     "."EMP2"
------------------------------------------------


CREATE TABLE "T1     "."EMP2"  (
               "EMPNO" CHAR(6) NOT NULL ,
               "FIRSTNME" VARCHAR(12) NOT NULL ,
               "MIDINIT" CHAR(1) ,
               "LASTNAME" VARCHAR(15) NOT NULL ,
               "WORKDEPT" CHAR(3) ,
               "PHONENO" CHAR(4) ,
               "HIREDATE" DATE ,
               "JOB" CHAR(8) ,
               "EDLEVEL" SMALLINT NOT NULL ,
               "SEX" CHAR(1) ,
               "BIRTHDATE" DATE ,
               "SALARY" DECIMAL(9,2) ,
               "BONUS" DECIMAL(9,2) ,
               "COMM" DECIMAL(9,2) )
              IN "IBMDB2SAMPLEREL" ;

COMMIT WORK;

CONNECT RESET;
TERMINATE;
```

注意：

db2look 命令非常强大，详细讲解请参见“第 13 章：数据库常用工具”。

5.2.13　表设计高级选项

除了上面介绍的一些特性外，DB2 还有很多高级特性，例如表分区、MDC 和表压缩等。

1. 多维集群(MDC)

多维集群允许物理地同时在多个键或维上将表集群。在 DB2 V8 之前，DB2 只支持使用集群索引的单维数据集群。在表上定义集群索引后，当在将记录插入表中或者更新表中的记录时，DB2 试图根据集群索引的键顺序维护数据在页上的物理顺序。对于那些具有包含集群索引的键的谓词的查询，这样可以大大提高性能，因为有了良好的集群之后，就只需要访问物理表的一部分。当页面按顺序存储在磁盘上时，预取的性能会更高。

有了 MDC，相同的优点被扩展到多个维或集群键上。在查询性能方面，涉及表中一个或多个指定维的范围查询将从底层的集群获得好处。这些查询只需要访问那些包含具有指定维值的记录的页，符合条件的页将组合在一起。随着时间的推移，当表中的可用空间被填满时，具有集群索引的表可能变为非集群的。然而，MDC 表可以自动、连续地在指定维上维护集群，而不必通过重组表来恢复数据的物理顺序。

当创建 MDC 表时，会指定用于顺着它们来集群表数据的维键。每个指定的维可以用一个或多个列来定义，这一点与索引键相同。对于每个指定的维，会自动创建维块索引，维块索引将用于快速、有效地沿着每个指定的维访问数据。此外，还会自动创建包含所有维键的块索引。块索引将用于维护插入和更新活动期间的数据集群，以及用于对数据进行快速有效的访问。

表的维值的每一种唯一组合都形成了逻辑单元，逻辑单元在物理上由一些页块组成，每个页块是磁盘上的一组连续的页。有一些页包含的数据在某个维块索引上具有相同键值，包含这些页的一组块称作切片(slice)。表的每个页只存储在单个块中，表的所有块由相同数量的页组成，即所谓的分块因子(blocking factor)。分块因子与表空间的盘区大小相等，因此块边界与盘区边界成线性关系。

例 5-15 创建 MDC 表。

为了创建 MDC 表，需要使用 organize by 参数指定表的维，如下所示：

```
CREATE TABLE MDCTABLE(
Year INT,
Nation CHAR(25),
Colour VARCHAR(10),  ... )
ORGANIZE BY(Year, Nation, Color)
```

在这个例子中，这个表将按 Year、Nation 和 Color 这几个维来组织，逻辑上看起来如图 5-11 所示。

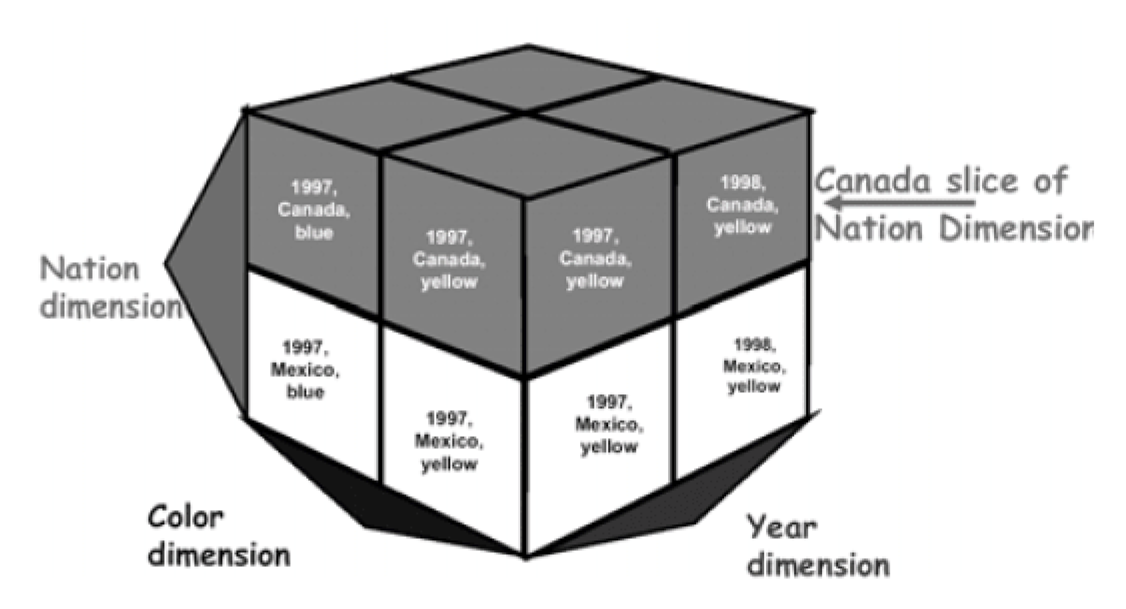


图 5-11　例 5-15 中 MDC 表的维逻辑示意图

不能将表改成 MDC 表，所以在创建数据库之前，应该尽可能根据业务需求来看看表应该是 MDC 表还是普通的表。关于 MDC 的详细介绍，请参见《DB2 数据库性能调整与优化(第 2 版)》中的“第 4 章：数据库物理设计和逻辑设计”。

2. 表(范围)分区

DB2 V8.2(及之前版本)的一些技术允许将数据拆分成更小的“簇”(cluster)，以获得更大的查询并行度，消除查询中出现的分区并帮助提高性能。正如前面所讨论的，MDC 允许 DB2 安排磁盘上的数据，使具有相同维列值的行在块(一组页)中存储在一起。通过使用这种技术，用于搜索具有特定维值的行的查询把所有其他分区排除在扫描之外，只有符合条件的行才会被访问。

类似地，数据库分区特性可以拆分一组表，使得一部分数据存放在分区上。数据库分区可以处于不同的服务器上，这样一来，大型的扫描可以使用多个服务器的处理能力。

DB2 V9 还引入了一种新形式的分区，该特性被称为表分区(table partitioning)，表分区允许将单个表扩展到多个表空间。

这种新的分区功能有很多优点，包括创建表的语法更简单。下面是简单的示例，创建的分区表用于将 24 个月的数据存储在 4 个表空间的 24 个分区中：

```
CREATE TABLE fact
(txn_id char(7), purchase_date date, ...)
                  IN tbsp1, tbsp2, tbsp3, tbsp4
                  PARTITION BY RANGE(purchase date)
(STARTING FROM('2005-01-01')
ENDING('2006-12-31')
EVERY1MONTH)
```

快速添加或删除数据范围

表分区的另一个优点是，当分离(detach)分区时，可以得到独立的表，这个独立的表包含了分区的内容。可以将分区从表中分离出来，然后对新分离出来的分区做一些处理，新分离出来的分区现在实际上是物理表。例如，可以归档那个表，将之移到第三存储，复制到另一个位置，或者做你想做的任何事情。DB2 V9 将异步地清除那个分区表上的任何索引键，而不影响正在运行的应用程序。

与添加新分区类似，只需要以和分区表相同的定义创建表，为之装入数据，然后将那个分区附加(attach)到主分区表上，如下所示：

```
ALTER TABLE FACT TABLE ATTACH PARTITION
STARTING '01-01-2007'
ENDING '01-31-2007'
FROM TABLE FACT_NEW_MONTH
```

关于表分区的详细介绍，请参见《DB2 数据库性能调整与优化(第 2 版)》一书的“第 4 章：数据库物理设计和逻辑设计”。

3. 表压缩

可以将压缩功能同时应用于新表和现有的表，也可以应用于临时表。要在 DB2 表中实现数据压缩，可以使用以下两种方法：

- 行压缩(需要具备 DB2 存储器优化功能部件的许可证才能获得此功能)
- 值压缩
- 自适应压缩(DB2 V10 开始支持)

表压缩的方法是查看整个表，找到重复的字符串，将那些字符串存储在字典中，然后用表示存储在字典中的实际数据的符号代替出现在表中的那些符号。主要优点是，DB2 看到的是表中的所有数据以及完整的数据行—— 而不只是重复的列值。例如，如果在列中有重复的子字符串，那么可以对列进行压缩。如果多个列中存在重复的字符串(例如城市、州、人的姓名等)，那么也可以将之压缩成单独的符号。

要使用表压缩，首先必须对表进行设置，使之可以被压缩，然后必须生成字典，字典中包含表中出现的重复字符串。要将表设置成可以被压缩，可以使用：

```
create table table_name ... compress yes
```

或

```
alter table tablename compress yes
```

创建压缩字典

创建压缩字典可以使表能够被压缩。DB2 需要扫描表中的数据，以发现表中出现的可以压缩的重复字符串，并将其放入字典中。为此，可以使用 reorg 命令。第一次压缩表(或是想重建压缩字典)时，必须运行命令：

```
reorg table  table_name resetdictionary
```

该命令将扫描表，创建字典，然后执行实际的表重组，从而压缩数据。此后，每当插入数据到表中或者为表装载数据时，都将遵从这个压缩字典，并压缩所有新的数据。如果将来想运行一次常规的表重组，但是不想重建这个字典，那么可以运行命令：

```
reorg table table_name keep dictionary
```

每个表都有自己的字典，这意味着对于每个分区，分区表都有单独的字典。这样很有好处，因为当卷入新的分区时，DB2 能够适应数据的变化。

估计压缩节省的空间

如果只是想看看能节省多少空间，而不想真正对表进行压缩，目前有两种方法：inspect 命令和表函数 admin_get_tab_compress_info。

使用 inspect：

```
db2 inspect rowcompestimate table name table_name results keep file_name
```

然后运行命令：

```
db2inspf file_name                    output_file_name
```

将二进制输出文件转换成名为 output_file_name 的文本文件，其中包含估计通过压缩可以节省的数据页的百分比。

使用表函数 admin_get_tab_compress_info：

```
db2 "create table tab like syscat.tables"
db2 "insert into tab select * from syscat.tables"
db2 "inspect rowcompestimate table name tab results keep tab_out "
db2inspf tab_out tab_out.log
```

输出如下：

```
DATABASE: SAMPLE
VERSION : SQL10010
2012-08-21-16.14.51.016844


Action: ROWCOMPESTIMATE TABLE
Schema name: DB2TEST1
Table name: TAB
Tablespace ID: 3  Object ID: 13
Result file name: tab_out

    Table phase start (ID Signed: 13, Unsigned: 13; Tablespace ID: 3) : DB2TEST1.TAB

      Data phase start. Object: 13  Tablespace: 3
      Row compression estimate results:
      Percentage of pages saved from compression: 81
      Percentage of bytes saved from compression: 81
      Compression dictionary size: 41216 bytes.
      Expansion dictionary size: 32768 bytes.
      Data phase end.
    Table phase end.
Processing has completed. 2012-08-21-16.14.51.068699
```

通过表函数可以得到同样的结果：

```
    db2 "select substr(tabschema,1,7) schema,substr(tabname,1,10)
tabname,compress_attr com,substr(dict_builder,1,10)
builder,dict_build_timestamp, substr(char(compress_dict_size),1,5)
dic_size,substr(char(expand_dict_size),1,5)
exp_size,rows_sampled,pages_saved_percent p_per,bytes_saved_percent
b_per,avg_compress_rec_length avg_com_len from
table(admin_get_tab_compress_info('CAOWW','TAB','estimate')) AS X"|more
```

```
SCHEMA  TABNAME    COM BUILDER    DICT_BUILD_TIMESTAMP       DIC_SIZE EXP_SIZE ROWS_SAMPLED P_PER  B_PER  AVG_COM_LEN
------- ---------- --- ---------- -------------------------- -------- -------- ------------ ------ ------ -----------
DB2TEST TAB        N   TABLE FUNC 2012-08-21-16.16.27.000000 41216    32768             480     81     81          79
```

对新表进行表压缩的步骤

如果从新表开始，那么可能需要：

(1) 用 compress yes 创建表。

(2) 将示例数据装载到表中。

(3) 用 resetdictionary 重组表以创建新的字典。

(4) 将剩下的数据装载到表中(这次的装载将遵从上述字典，并在装载的同时进行压缩)。

关于表压缩的详细介绍，请参见《DB2 数据库性能调整与优化(第 2 版)》中的“第 4 章：数据库物理设计和逻辑设计”。

5.3　索引设计

5.3.1　索引的优点

索引是表的一个或多个列的键值的有序列表。创建索引的原因有两个：

- 确保一个或多个列中值的唯一性。
- 提高表的查询性能。当执行查询想以更快的速度找到所需的列时，或要以索引的顺序显示查询结果时，DB2 优化器选择使用索引。如果表上不存在索引，那么必须对 SQL 查询中引用的每个表执行表扫描。表越大，表扫描所花的时间越长，因为表扫描需要顺序访问每个表行。虽然对于需要表中大多数行的复杂查询来说，使用表扫描效率可能更高，但是对于只返回部分表行的查询而言，使用索引扫描可以更有效地访问表行。

如果在 SELECT 语句中引用了索引列，并且优化器估计索引扫描比表扫描快，那么优化器选择索引扫描。索引文件一般较小，因此读取所需的时间比读取整个表所需的时间要少，尤其在表增大时更是如此。此外，可能不需要扫描整个索引。应用于索引的谓词减少了要从数据页读取的行数。

如果对输出的排序需求可以与索引列相匹配，那么按列顺序扫描索引将允许按正确顺序检索行而不需要排序。

每个索引条目包含一个搜索键值和一个指向包含该值的行的指针。如果在 CREATE INDEX 语句中指定 ALLOW REVERSE SCANS 参数，那么可以按升序和降序搜索这些值。因此，在具有正确谓词的情况下，才可能对搜索分类。也可使用索引来获得已排序的行，使数据库管理器在从表中读取这些行之后不必对它们排序。

除搜索键值和行指针外，索引还可包含(include)列，这些列是索引行中的非索引列，但是它们的数据包含在索引叶子中。这样的列有可能使优化器仅从索引获取所需要的信息，而不必访问表本身。关于 include 索引，下面有详细讲解。

注意：

要查询的表上存在索引并不保证结果集已排序。只有 ORDER BY 子句能确保结果集的排序。

尽管索引可显著缩短访问时间，但是它们也可给性能带来负面影响。在创建索引之前，考虑多个索引给磁盘空间和处理时间带来的影响：

- 每个索引都需要存储器或磁盘空间。准确的容量取决于表的大小以及关系索引中列的大小和数目。

- 对表执行的每个 INSERT 或 DELETE 操作都需要对表上的每个索引进行额外的更新，对于更改索引键值的每个 UPDATE 操作也是如此。
- LOAD 实用程序重建任何现有的关系索引或追加至现有的关系索引。可在 LOAD 命令中指定 index freespace MODIFIED BY 参数，以覆盖创建索引时使用的索引 PCTFREE。每个关系索引都有可能对 SQL 查询添加备用访问路径以供优化器考虑，这会增加编译时间。

因此，需要谨慎选择索引来满足应用程序的需要。

注意：

关系索引是相对于 XML 索引而言的，所以关系索引就是常规索引，也就是通常意义上的索引。在 DB2 V9 之前关系索引就是索引，DB2 V9 中有 XML 索引，为了加以区分，又称常规索引为关系索引。

5.3.2 索引类型

有如下几种类型的索引：唯一索引、非唯一索引、集群索引、非集群索引、分区和非分区索引以及系统为多维集群(MDC)表生成的块索引。

唯一索引和非唯一索引

唯一索引是这样一种索引，通过确保表中没有两个数据行具有完全相同的键值来帮助维护数据完整性。

尝试为已经包含数据的表创建唯一索引时，将检查组成唯一索引的列中的值是否唯一；如果表中包含具有重复键值的行，那么索引创建过程将失败。为表定义了唯一索引之后，每当在索引中添加或更改键时就会强制唯一性(这包括插入、更新、装入、导入和设置完整性以命名一部分)。除了强制数据值的唯一性以外，唯一索引还可用来提高查询处理期间检索数据的性能。

另一方面，非唯一索引不用于对与它们关联的表强制执行约束。相反，非唯一索引维护频繁使用的数据值的排序顺序，这仅仅用于提高查询性能。

集群索引和非集群索引

索引体系结构分为集群或非集群。集群索引是这样一种索引，数据页中行的顺序对应于索引中行的顺序。这就是为何给定表中只能存在一个集群索引，而表中可以存在多个非集群索引的原因。在某些关系数据库管理系统中，集群索引的叶子节点对应于实际数据，而不是对应于指向位于其他地方的数据的指针。图 5-12 是集群索引和非集群索引的示意图。

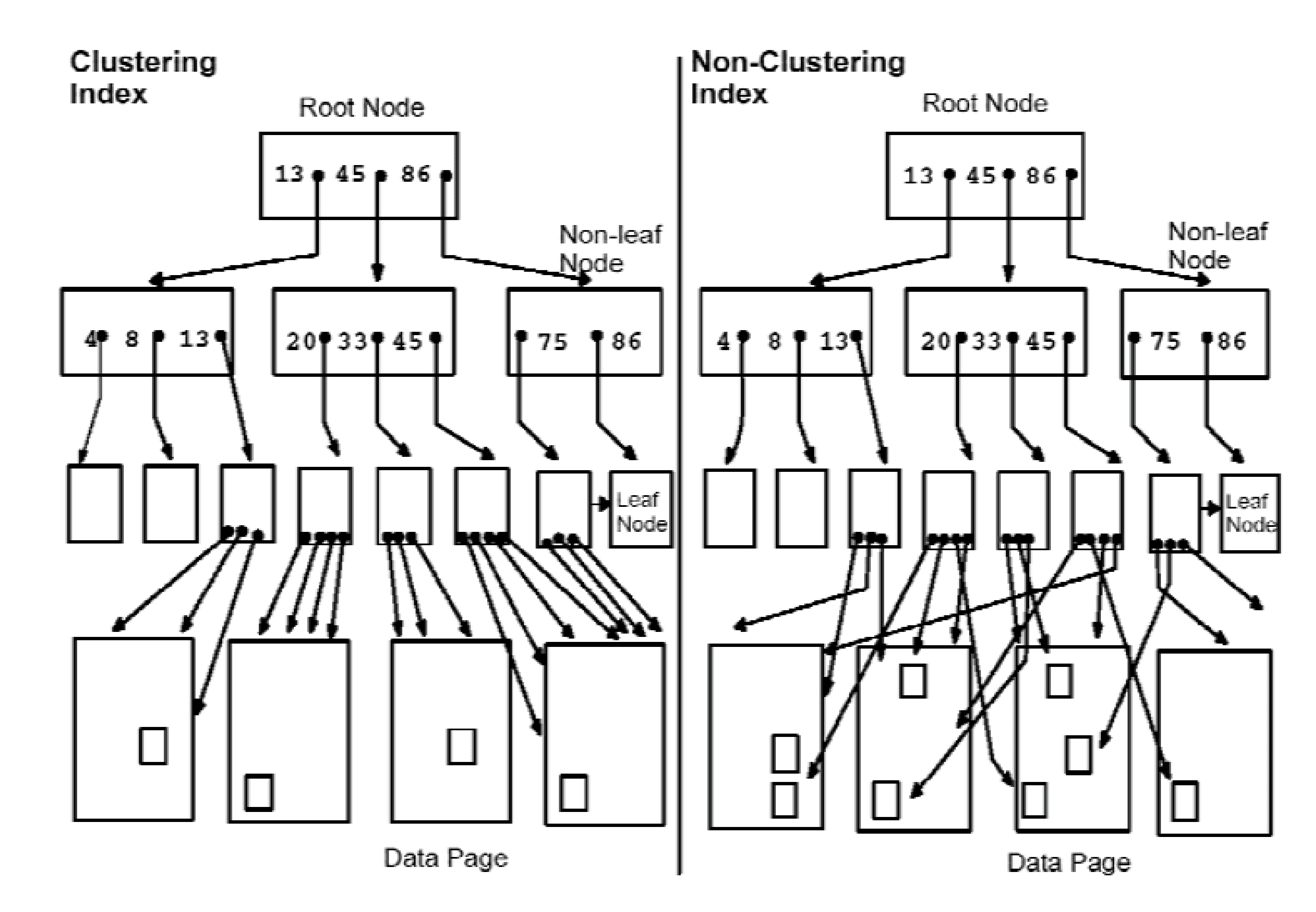


图 5-12　集群索引和非集群索引

集群索引和非集群索引都只包含索引结构中的键和记录标识。记录标识始终指向数据页中的行。集群索引和非集群索引的区别在于：数据库管理器尝试按照相应的键在索引页中的出现顺序来将数据保存在数据页中。因此，数据库管理器将尝试把具有相似键的行插入同一页中。如果对表进行重组，那么会按照索引键的顺序将行插入数据页中。

通常，表中只有一个索引可以具有较高的集群度(因为实际的数据只可能有一种物理存放顺序)。集群索引改善了以键的顺序扫描整张表的性能。这是因为首先扫描访存第一页的第一行，然后访存同一页上的每一行，在访存了该页的所有行之后，才移至下一页。这意味着任何给定时间内都只需要表的一页位于缓冲池中。相反，如果表未集群，那么访存的每行有可能是在不同的页中。除非缓冲池中有空间保存整个表，否则这会导致每页被访存多次，从而极大地减慢扫描速度。

主键和唯一键约束与唯一索引之间的差别

了解主键和唯一键约束与唯一索引之间没有很大差别这一点很重要。数据库管理器使用唯一索引和 NOT NULL 约束的组合来实现主键约束和唯一键约束。因此，唯一索引本身不强制执行主键约束，因为它们允许空值(虽然空值表示未知值，但在建立索引时，会将一个空值视为与其他空值相同)。

因此，如果唯一索引由单个列组成，那么只允许一个空值，多个空值将违反唯一约束。同样，如果唯一索引由多个列组成，那么值和空值的特定组合只能使用一次。

双向索引

默认情况下，双向索引允许按正反两个方向进行扫描。CREATE INDEX 语句的 ALLOW REVERSE SCANS 子句同时启用正向和反向索引扫描，也就是说，按创建索引时的顺序和相反(或反向)顺序。此选项使得：

- 便于使用 MIN 和 MAX 函数
- 访存先前的键
- 不需要数据库管理器创建临时表来进行反向扫描
- 消除冗余反向顺序索引

如果指定 DISALLOW REVERSE SCANS，那么不能反向扫描索引。

MDC 块索引

在创建 MDC 时，数据库会自动生成块(block)索引，它和常规行索引的区别如图 5-13 所示。

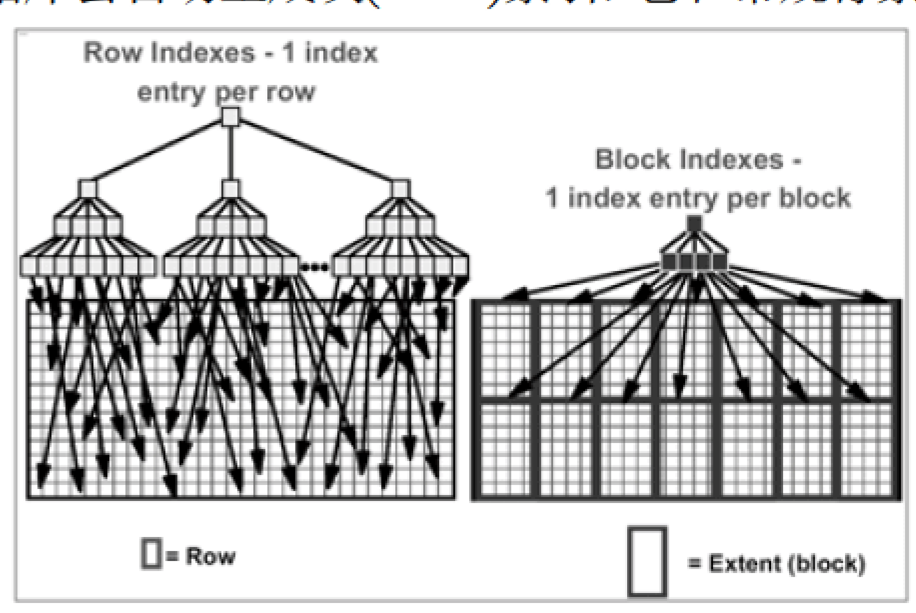


图 5-13　MDC 块索引与常规行索引的区别

从上面的图 5-13 中可以看到，MDC 索引相关的索引块是排列到一起的，所以可以显著地提高性能。这部分内容本书中不做详细讲解，我们会在《高级进阶 DB2(第 2 版)》一书中深入介绍。

分区索引和非分区索引

非分区索引是单一的索引对象，可以引用分区表中所有的行。非分区索引始终作为单一表空间中的独立索引对象进行创建，即使表数据分区跨多个表空间也是如此。

当为分区表创建索引时，索引在默认情况下为分区索引，除非创建下列其中一种类型的索引：

- 索引键未包含所有表分区列的唯一索引。
- 在 XML 数据列上创建的唯一索引。
- 在 spatial 数据列上创建索引。

- 在 XML 列路径上创建索引。

空间索引

分区索引由一组索引分区构成，每个索引分区都包含单一数据分区的索引条目。每个索引分区都只包含对相应数据分区中数据的引用。系统生成的索引和用户生成的索引都可以是分区索引。

5.3.3　索引结构

标准表的表和索引管理

在图 5-14 中，我们可以看到在标准表中，数据在逻辑上是按数据页的列表来组织的。这些数据页根据表空间的扩展数据块大小在逻辑上分组在一起。例如，如果扩展数据块(extent)大小是 4，第 0 至 3 页是第一个扩展数据块的一部分；那么第 4 至 7 页就是第二个扩展数据块的一部分，依此类推。

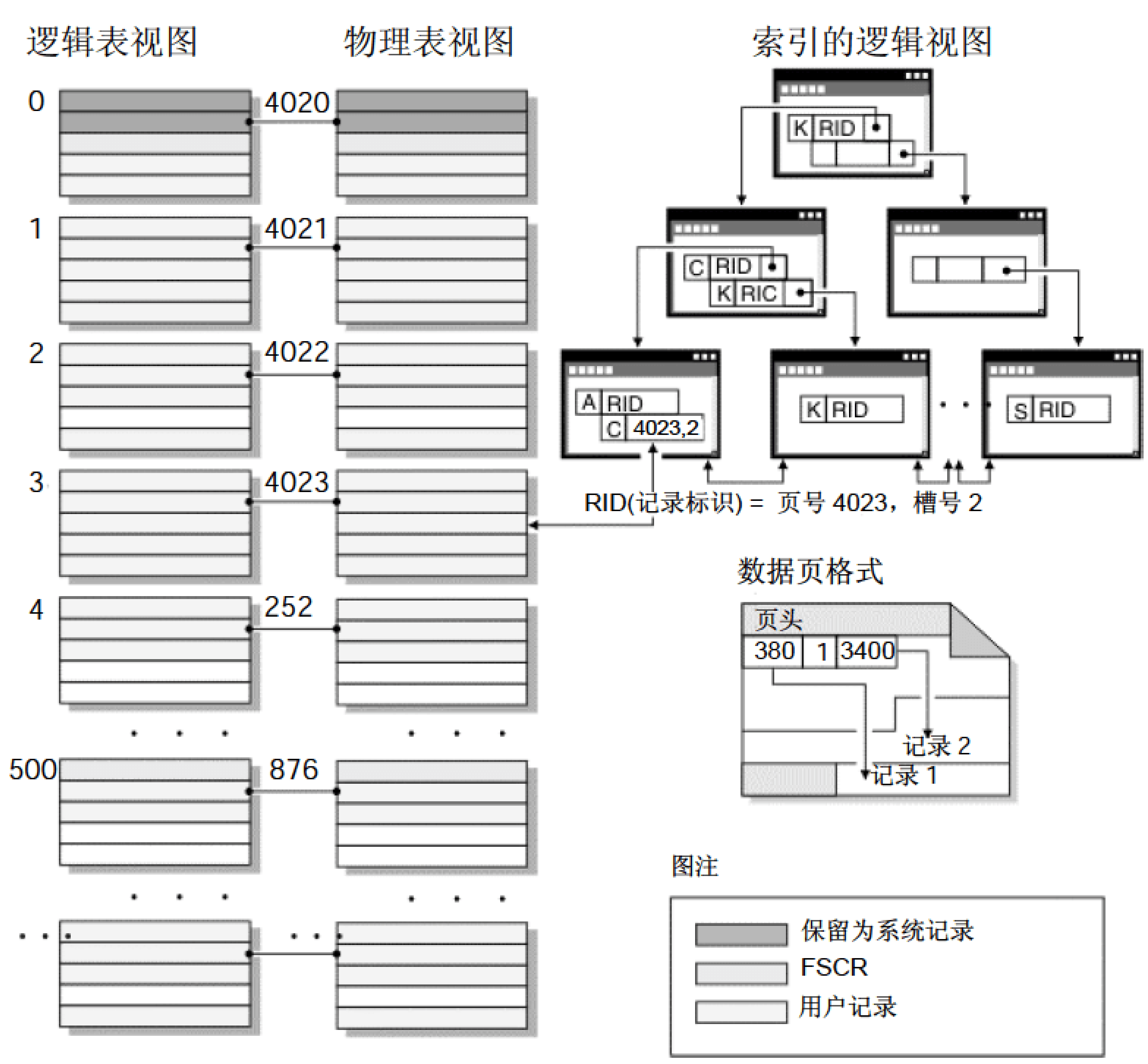


图 5-14　标准表的逻辑表、记录和索引结构

根据数据页大小以及记录大小的不同，每个数据页中包含的记录数可能会有所变化。大多数页仅包含用户记录。但是，少数页包括特殊的内部记录，DB2 使用这些记录来管理表。例如，在标准表中，每个第 500 个数据页上都有“空闲空间控制记录”(FSCR)。这些记录映射下面每 500 个数据页(直到下一 FSCR 为止)上的可供新记录使用的可用空间。当将记录插入表中时，将使用这部分可用空间。

在逻辑上，索引页组织成 B+树，这可以有效地在表中定位带有给定键值的记录。索引页上的项数不是固定的，但依赖于键的大小。对于 DMS 表空间中的表来说，索引页中的记录标识(RID)使用相对表空间页号，而不是对象相对页号。这使索引扫描能够直接访问数据页，而不需要“扩展数据块映像页”(EMP)来进行映射。

每个数据页都具有相同的格式。每个数据页的开头都有页头。在页头后面，有槽目录。槽目录中的每一条目都与数据页中的某个记录相对应。条目本身是数据页中记录开始位置的字节位移。值为－1 的条目与已删除的记录相对应。

记录标识和页

记录标识(RID)由页号及随后的槽号组成。DB2 V8 后的 Type 2 索引记录还包含称为 ridFlag 的附加字段。该字段存储有关索引中密钥状态的信息，例如此密钥是否标记为已删除。在使用索引标识了 RID 之后，便可以使用 RID 来到达正确的数据页以及数据页上正确的槽号。一旦对记录指定 RID，在进行表重组之前，RID 便不会更改。数据页和 RID 的格式如图 5-15 所示。

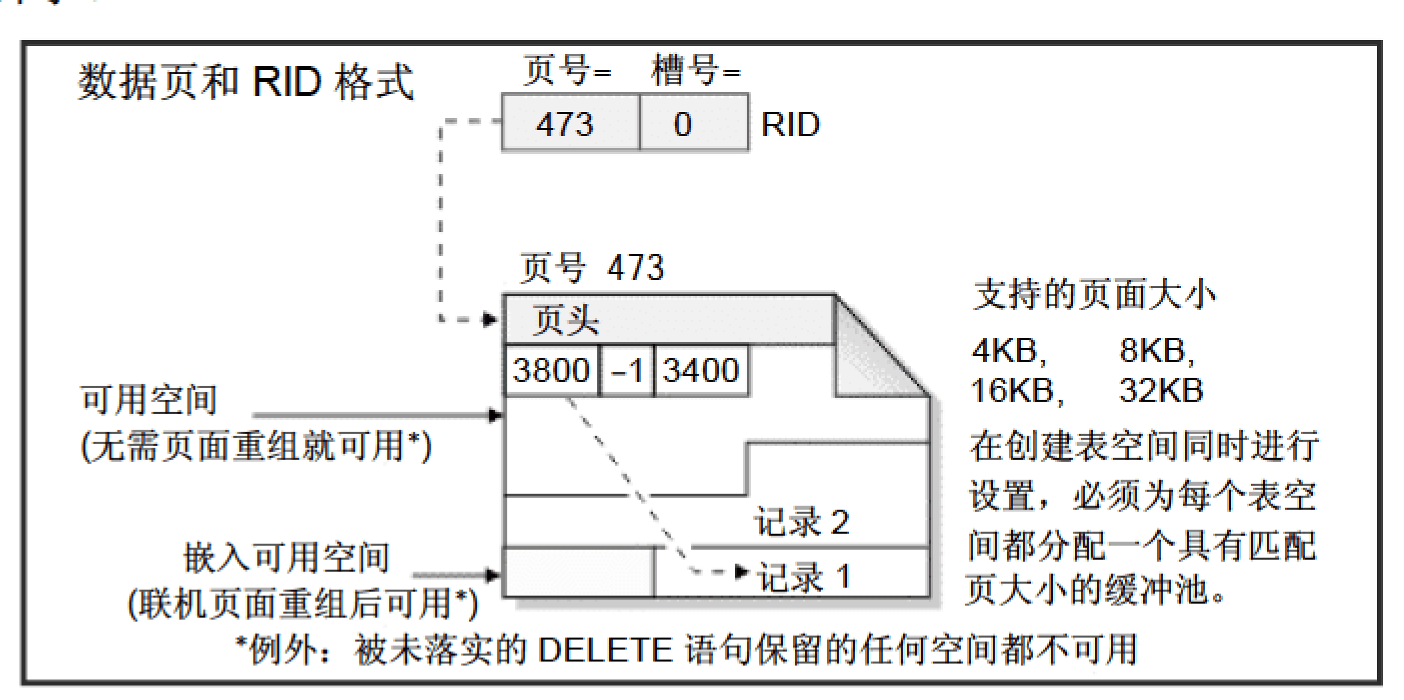


图 5-15　数据页和记录标识的格式

重组表时，实际删除记录后，在数据页上留下的嵌入可用空间被转换成可使用的可用空间。根据记录在数据页上的移动重新定义 RID，以利用可使用的可用空间。

DB2 支持不同的页大小。对于有可能连续访问行的工作负载，请使用较大的页大小。例如，“决策支持”应用程序或大量使用临时表的场合使用的便是顺序访问。对于更有可能

进行随机访问的工作负载，使用较小的页大小。例如，OLTP 环境中使用的便是随机访问。

B+树结构

DB2 数据库管理器使用 B+树结构(Oracle 数据库有位图索引，DB2 没有位图索引)进行索引存储。B+树有一层或多层，如图 5-16 所示。其中，RID 表示行标识。

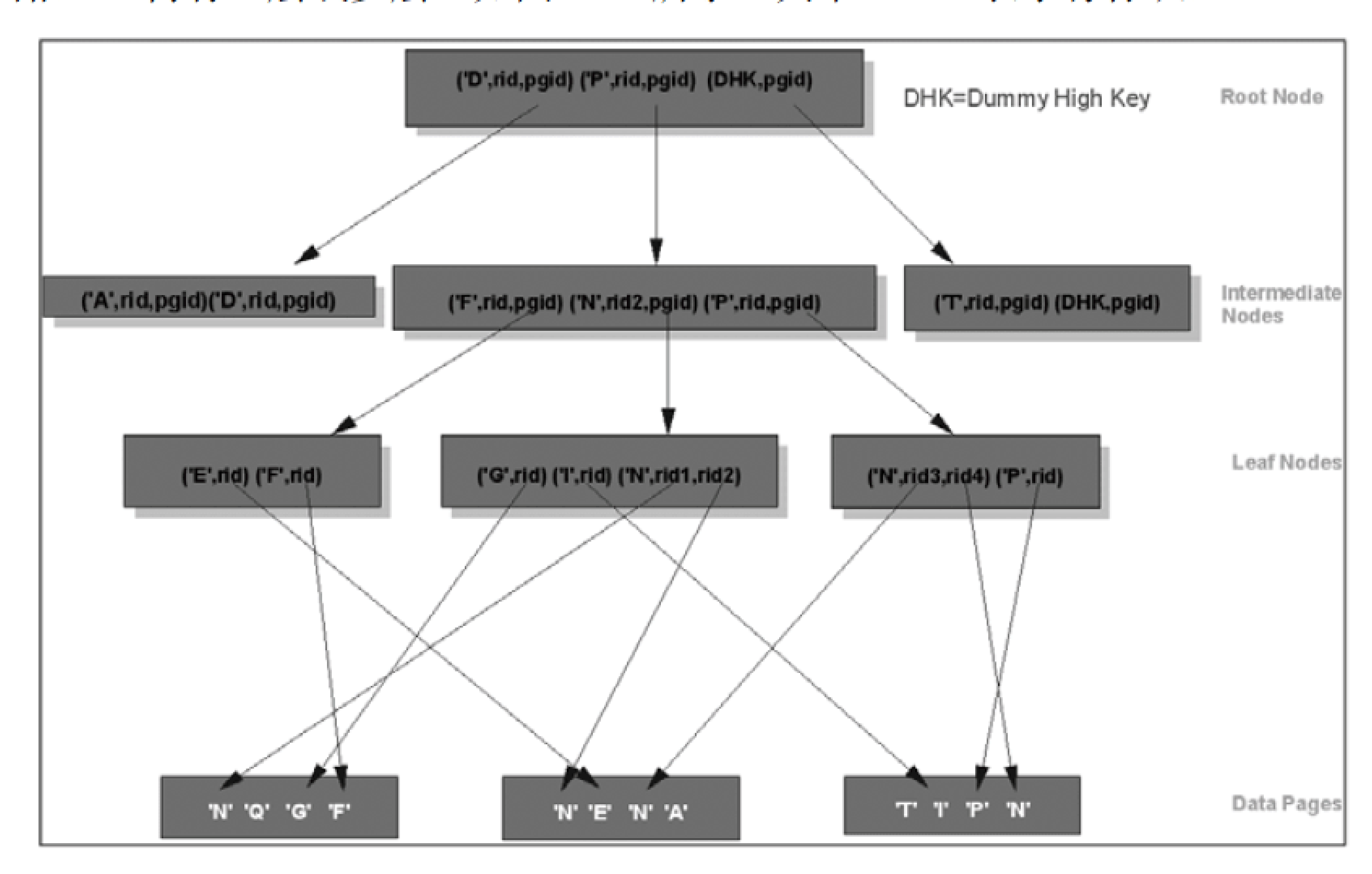


图 5-16　B+树结构

顶层称为根节点。底层由叶节点组成。底层存储了索引键值，并有指针(RID)指向包含键值的表中的行。根节点层和叶节点层之间的那些层称为中间节点。

当查找特定的索引键值时，索引管理器会从根节点开始搜索索引树。对于下一层的每个节点根都包含键，每个键的值是下一层中对应节点的最大现有键值。例如，如果索引有 3 层(如图 5-16 所示)，那么要查找索引键值，索引管理器会搜索根节点以查找大于或等于要查找键的第一个键值；根节点键指向特定的中间节点。索引管理器遵循此过程遍历中间节点，直到找到包含所需要的索引键的叶节点为止。

5.3.4　理解索引的访问机制

在优化器必须做的许多决定中，最重要的决定可能是——是否使用索引来实现查询。在优化器做此项决定之前，必须首先确定是否存在索引。请记住，可以查询任何表中的任何列，却不能期望单单通过索引就能做到这一点。所以，优化器必须能够访问未建立索引的数据；可以使用扫描来做到这一点。

在大多数情形下，DB2 优化器喜欢使用索引。这是事实，因为索引可以大大优化数据检索。然而，如果不存在索引，就无法使用。并且在某些情况下仅仅使用数据的全扫描就可以极好地实现某些类型的 SQL 语句。例如，考虑下面这条 SQL 语句：

```
SELECT * FROM EMP;
```

在这条语句中，为什么 DB2 非要试图使用索引呢？这里没有 WHERE 子句，所以全扫描是最佳的。即使指定 WHERE 子句，如果优化器确定页面的全表扫描要比索引扫描更好，那么也可能不会选择索引扫描这种方法。

存在索引的首要原因是可以改善性能，那为什么有时全表扫描会比索引扫描还要好呢？这是因为索引扫描可能比简单的表扫描要慢。例如，一个非常小的表可能只有几个页面。读取所有的页面可能比先读取索引页，然后再读取数据页要快。甚至对于较大的表，在某些情况下，组织索引可能需要额外的 I/O 以实现查询。当不使用索引来实现查询时，产生的存取路径会采用表扫描方法。

表扫描通常会读取表中的每个页面。但在某些情况下，DB2 会非常聪明，它会限定要扫描的页面。此外，DB2 可以调用顺序预取以在请求某些页面之前就读取这些页面。当 SQL 请求需要按照数据存储在磁盘上的顺序来顺序地访问多行数据时，顺序预取特别有用。当优化器确定查询将按照顺序读取数据页面时，它会通知应该启用顺序预取。表扫描常常得益于顺序预取所做的提前预取工作，因为当某个查询请求数据时，这些数据已经放在内存中。

快速的索引式访问

一般来讲，访问 DB2 数据的最快方式是使用索引。索引是为了能够快速找到某个特定数据块而构造的。图 5-16 显示了 B+树索引的结构。可以看到，通过简单地从树根遍历到叶子页，可以快速地找到相应的数据页，在那里有你请求的数据。但是，DB2 采用的索引方式因语句不同而各不相同。DB2 使用各种不同的内部算法来遍历索引结构。

在 DB2 使用索引来实现数据访问请求之前，必须满足以下条件：

- 至少有一个 SQL 谓词必须是可索引的。某些谓词因特性而不能被索引，所以优化器不能使用索引来满足它们。
- 其中一列(在任何可索引谓词中)必须作为可用索引中的列而存在。

所以，对于 DB2，考虑使用索引的要求是相当简单的。但关于 DB2 中的索引扫描，仍有许多要了解的内容。事实上，索引扫描有多种类型。

直接索引查找(也是最简单的)

对于直接索引查找，DB2 使用索引的根页面，从顶部开始，向下遍历，经过中间叶子页，直到抵达相应的叶子页。在那里，将读取实际数据页面的指针。根据索引条目，DB2 将读取正确的数据页以返回期望的结果。对于 DB2 而言，为了执行直接索引查找，在查询的 WHERE 子句中必须为每个列提供值。例如，考虑 EMPLOYEE 表，该表有关于 DEPTNO、TYPE 和 EMPCODE 列的索引。现在考虑下面这个查询：

```
SELECT   FIRSTNAME, LASTNAMEF  ROM     EMPLOYEE
WHERE    DEPTNO = 5 AND      TYPE = 'X'   AND      EMPCODE = 10;
```

如果只指定这些列中的一列或两列，就不可能采用直接索引查找，因为 DB2 没有针对每列的值，不可能匹配整个索引关键字。相反，可以选用索引扫描。有两类索引扫描：匹配索引扫描和非匹配索引扫描。

有时称匹配索引扫描为绝对定位；称非匹配索引扫描为相对定位。还记得前面讨论的表扫描吗？索引扫描与此类似。在索引扫描中，按顺序读取索引的叶子页。

匹配索引扫描从索引的根页开始，遍历至叶子页，这种扫描方式与直接索引查找方式完全一样。然而，因为无法用完整的索引关键字，所以 DB2 必须使用拥有的值来扫描叶子页，直到检索出所有匹配的值。现在考虑重写前面那个查询，这次没有使用 EMPCODE 谓词：

```
SELECT   FIRSTNAME, LASTNAME  FROM     EMPLOYEE
WHERE    DEPTNO = 5 AND      TYPE = 'X';
```

通过从根部开始遍历索引，匹配索引扫描用相应的 DEPTNO 和 TYPE 值来查找第一个叶子页。但可能有多条索引条目具有这两个值的组合，而这些索引条目的 EMPCODE 值却不同。所以，会按顺序扫描至右边的叶子页，直到不再遇到有效的 DEPTNO、TYPE 和各种 EMPCODE 的组合。

要请求执行匹配索引，必须指定索引关键字中的高次序列，就是前面这个示例中的 DEPTNO。这向 DB2 提供了遍历索引结构的起始点，从根页开始遍历，直到相应的叶子页。但如果没有指定这个高次序列，那么会发生什么呢？假定对上面这个样本查询做点改动，不指定 DEPTNO 谓词：

```
SELECT   FIRSTNAME,  LASTNAME  FROM     EMPLOYEE
WHERE    TYPE = 'X'  AND      EMPCODE = 10;
```

在这个示例中，会用到非匹配索引扫描。在这种情形下，DB2 不能使用索引树结构，因为关键字中的第一列不可用。非匹配索引扫描从索引中的第一个叶子页开始遍历，应用

可用的谓词，顺序扫描后续的叶子页。不使用根页和任何中间叶子页。

完全索引访问(index access only)

还有一种特殊类型的索引扫描，就是“完全索引访问(index access only)”。如果需要的全部数据都位于索引中，那么 DB2 完全可以避免读取数据页。例如：

```
SELECT   DEPTNO, TYPE
FROM     EMPLOYEE
WHERE    EMPCODE = 10;
```

请记住，这个数据库中包含 DEPTNO、TYPE 和 EMPCODE 列的索引。在前面的查询中，只请求查询这几列。所以，DB2 完全不需要访问表，因为在索引中可以找到所有数据。

多索引访问

DB2 可使用的另一类索引访问是多索引访问。针对存取路径，多索引访问将使用多个索引。例如，查询 EMPLOYEE 表，其中只有两个索引：关于 EMPNO 列的 IX1 和关于 DEPTNO 列的 IX2。然后，要求这条查询显示在某个特定部门工作的员工：

```
SELECT   LASTNAME, FIRSTNME, MIDINIT
FROM     EMPLOYEE
WHERE    EMPNO IN('000100', '000110', '000120')
AND      DEPTNO = 5;
```

DB2 将会使用用于 EMPNO 谓词的 IX1 还是使用用于 DEPTNO 谓词的 IX2？为什么不一起使用这两者呢？这就是多索引访问的实质所在。根据谓词是用 AND 连接还是用 OR 连接，可将多索引访问分为两类：IndexANDING 和 IndexORING。IndexANDING 是先使用索引 IX1 和 IX2 获取到索引扫描的结果，然后再对两个扫描结果取交集；而 IndexORING 是在使用索引 IX1 和 IX2 取得索引扫描结果后，执行合并操作。

5.3.5　创建集群索引

以下 SQL 语句在 EMPLOYEE 表的 LASTNAME 列上创建群集索引，名为 INDEX1：

```
CREATE INDEX INDEX1 ON EMPLOYEE(LASTNAME) CLUSTER
```

为了让语句更有效，可以通过与 ALTER TABLE 语句相关的 PCTFREE 参数来使用群集索引，以便将新数据插入正确的页中，从而维护该群集的次序。通常情况下，表上的 INSERT 操作越多，为维护群集所需要的 PCTFREE 值就越大。因为这个索引确定数据在物理页上放置的次序，所以对任何特定的表只能定义一个群集索引。

另一方面，如果这些新行的索引关键字值总是新的大关键字值，那么表的群集属性将尝试把它们放到表的末尾。其他页上有空闲空间对保持群集没有什么作用。在这种情况下，将表设置为追加方式可能优于使用群集索引，改变表以拥有大的 PCTFREE 值。可以执行如下命令来将表设置为追加方式：ALTER TABLE APPEND ON。

以上讨论也适用于增大行大小的 UPDATE 引起的新的“溢出”(overflow)行。

5.3.6　创建双向索引

使用 CREATE INDEX 语句中的 ALLOW REVERSE SCANS 参数创建的单索引可以向左或者向右扫描。也就是说，这些索引支持按照在反方向创建和扫描索引时定义的方向索引。这条 SQL 语句如下：

```
CREATE INDEX iname ON tname(cname DESC) ALLOW REVERSE SCANS
```

在这种情况下，基于给定列(cname)中的递减值(DESC)形成索引(iname)。尽管列上的索引定义用来按照递减次序扫描，通过允许反向扫描，可以按照降序(反向)扫描。实际上没有使用这两个方向上的索引，创建和考虑存取模式时由优化器控制这些索引的使用。

索引页的合并与分裂

CREATE INDEX 语句的 MINPCTUSED 子句指定在索引页上最小已用空间的阈值。如果使用这条子句，那么可以对这个索引启用联机索引重组。一旦启用联机索引重组，就可以参照以下考虑事项来确定是否执行联机重组：当从这个索引的索引页(leaf)中删除关键字(key)后，如果索引页上已用空间的百分比小于指定的阈值，那么就检查相邻的索引页来确定是否可以将两个索引页上的关键字合并到单个索引页中。例如，下列 SQL 语句创建启用联机索引重组的索引：

```
CREATE INDEX LASTN ON EMPLOYEE(LASTNAME) MINPCTUSED 20
```

当从这个索引删除关键字时，如果这个索引页上的其余关键字占用索引页上 20%或更小的空间，那么就可以尝试将这个索引页的关键字与相邻索引页的关键字合并，从而删除这个索引页。如果组合的关键字可以全部位于一页上，那么就执行这个合并，并删除其中一个索引页。

图 5-17 是设置了 MINPCTUSED 后索引在线重组的示意图。

CREATE INDEX 语句的 PCTFREE 子句指定，创建索引时每个索引页中要留作空闲空间的百分比。在索引页上保留更多的空闲空间将导致更少的页分割，这将减少为重新获得顺序索引页而重组表的需要，从而增加预存取，而预存取是可以提高性能的重要部件。此外，如果总是存在大关键字值，那么就要考虑降低 CREATE INDEX 语句的 PCTFREE 子句

的值。

对于只读表上的索引，使 PCTFREE 为 0；对于其他索引，使 PCTFREE 为 10 以提供可用空间，从而加快插入操作的速度。此外，对于有群集索引的表而言，这个值应该更大一些，以确保群集索引不会被分成太多的碎片。如果存在大量的插入操作，那么使用 15 到 35 之间的值或许更合适一些。

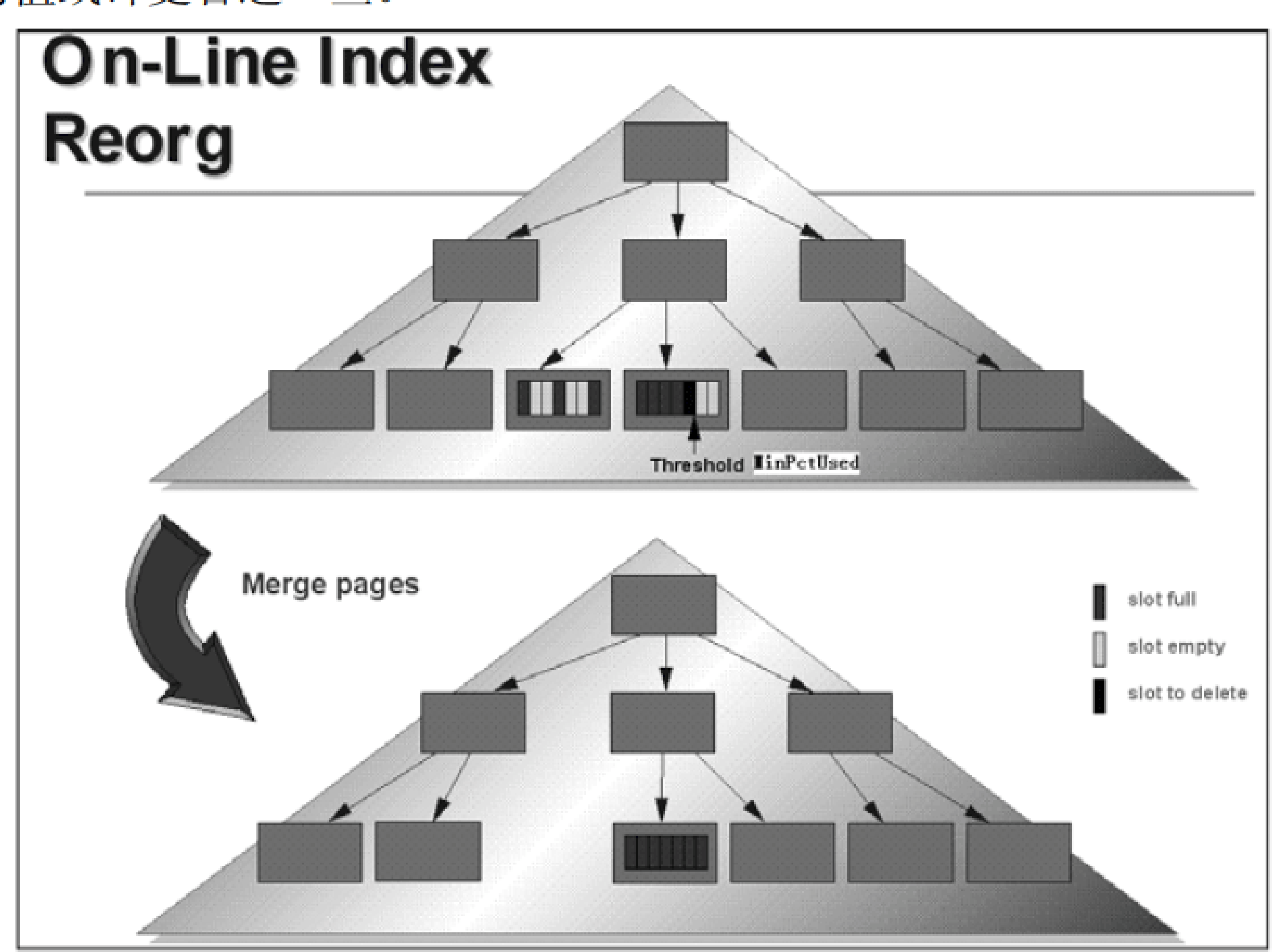


图 5-17　索引的在线重组

5.3.7　完全索引访问(index access only)

CREATE INDEX 语句的 INCLUDE 子句指定在创建索引时，可以选择包含附加的列数据，这些附加的列数据将与键存储在一起，但实际上它们不是键自身的一部分，所以不被排序。在索引中包含附加列的主要原因是为了提高某些查询的性能。DB2 不需要访问数据页，因为索引页早已经提供了数据值。只可以为包含的列定义唯一索引。但在强制执行索引的唯一性时不考虑被包含的列。

假设经常需要获得按 EMPNO 排序的员工列表。查询将如下所示：

```
SELECT EMPNO,EMPNAME FROM EMP ORDER BY EMPNO
```

下面的语句会创建可以提高性能的可能索引：

```
CREATE UNIQUE INDEX IEMPNO ON EMPNO(EMPNO) INCLUDE(EMPNAME)
```

结果，查询结果所需的所有数据都显示在索引中，不需要检索数据页。那么，为什么

不干脆在索引中包括所有的数据呢？首先，这需要数据库中的更多物理空间，因为本质上数据是在索引中复制的。其次，只要更新了数据的值，数据的所有副本就都需要更新，在发生许多次更新的数据库中，这是一项很大的开销。

5.3.8 创建索引示例

在最频繁处理的查询和事务的 WHERE 子句中使用的那些列上创建关系索引。

以下 WHERE 子句：

```
WHERE WORKDEPT='A01' OR WORKDEPT='E21'
```

通常会从 WORKDEPT 上的索引获益，除非 WORKDEPT 列包含许多重复值。

在按查询需要的顺序对行排序的一列或多列上创建关系索引时，不仅在 ORDER BY 子句中，而且其他功能，比如 DISTINCT 和 GROUP BY 子句也都需要排序。

以下示例使用 DISTINCT 子句：

```
SELECT DISTINCT WORKDEPT          FROM EMPLOYEE
```

数据库管理器可使用 WORKDEPT 上定义为升序或降序的索引来消除重复值，此索引也可用于 GROUP BY 子句中，从而将值分组，如下所示：

```
 SELECT WORKDEPT, AVERAGE(SALARY)
 FROM EMPLOYEE      GROUP BY WORKDEPT
```

使用复合键创建索引，该键命名语句中引用的每个列。当用此方式指定索引时，可以从完全索引检索关系数据，这比访问表更有效。

例如，考虑下列 SQL 语句：

```
SELECT LASTNAME   FROM EMPLOYEE  WHERE WORKDEPT IN('A00','D11','D21')
```

如果为 EMPLOYEE 表的 WORKDEPT 和 LASTNAME 列定义关系索引，那么通过扫描索引而不是扫描整个表，可能会更有效地处理该语句。注意，因为该谓词基于 WORKDEPT，因此该列应是关系索引的第一列。

使用 INCLUDE 列创建关系索引可改善表上索引的使用。使用上述示例，可将唯一关系索引定义为：

```
CREATE UNIQUE INDEX x ON employee(workdept) INCLUDE(lastname)
```

指定 lastname 为 INCLUDE 列而不是索引键的一部分，意味着 lastname 只存储在索引的叶子页上。

1. 根据查询使用的列建立索引

建立索引是用来提高查询性能的最常用方法。对于特定的查询，可以为某个表出现在查询中的所有列建立复合索引，包括出现在 SELECT 语句和条件子句中的列。但简单地建立覆盖所有列的索引并不一定能有效提高查询，因为在多列索引中列的顺序是非常重要的。这个特性是由索引的 B+树结构决定的。一般情况下，要根据谓词的选择度来排列索引中各列的位置，选择度大的谓词使用的列放在索引的前面，把那些只存在于 SELECT 语句中的列放在索引的最后。例如下面的查询：

例 5-16 索引中的谓词位置。

```
select add date  from temp.customer  where city = 'WASHINGTON'
and cntry_code = 'USA';
```

对于这样的查询，可以在 temp.customer 上建立(city，cntry_code，add_date)索引。由于该索引包含了 temp.customer 用到的所有列，因此该查询将不会访问 temp.customer 的数据页，而是直接使用索引页。对于包含多列的复合索引，索引树中的根节点和中间节点存储了多列的值的联合。这就决定了存在两种索引扫描。回到例 5-15 中的查询，由于此查询在新建索引的第一列上存在谓词条件，因此 DB2 能够根据这个谓词条件从索引树的根节点开始遍历，经过中间节点，最后定位到某个叶子节点，然后从此叶子节点开始往后进行叶子节点上的索引扫描，直到找到所有满足条件的记录。这种索引扫描就是我们前面所讲的匹配索引扫描(Matching Index Scan)。但是如果将 add_date 放在索引的第一个位置，而查询并不存在 add_date 上的谓词条件，那么这个索引扫描将会从第一个索引叶子节点开始，但无法从根节点开始并经过中间节点直接定位到某个叶子节点，这种扫描的范围扩大到了整个索引，这就是前面提到的非匹配索引扫描(Non-Matching Index Scan)。图 5-18 显示了 DB2 根据不同索引生成的存取计划。

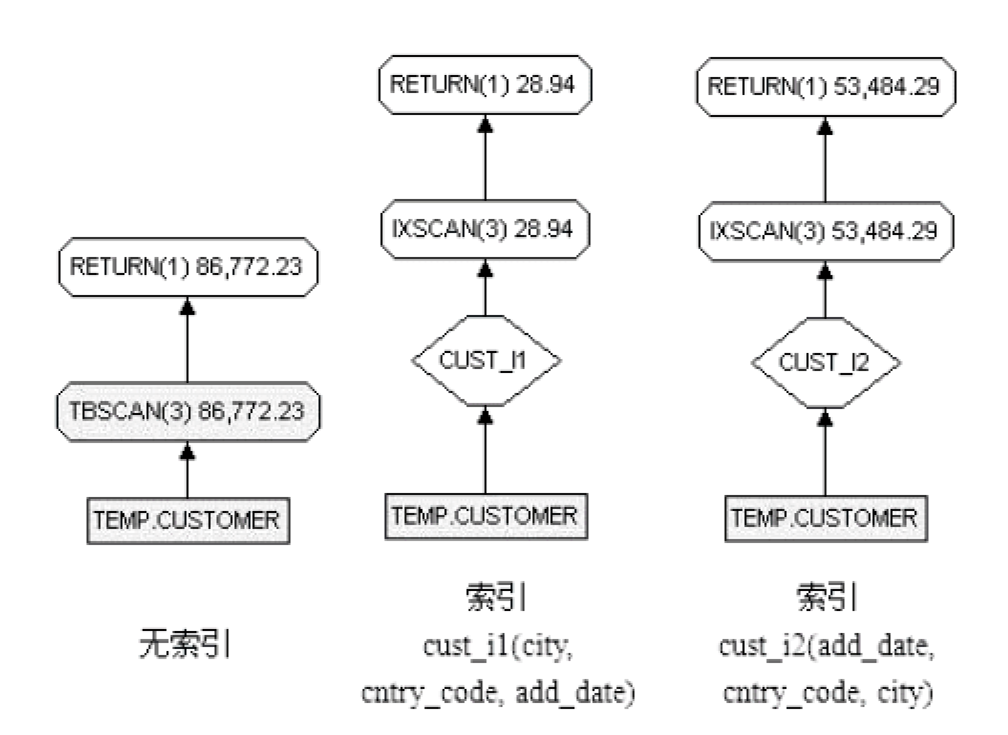


图 5-18 根据不同索引生成的存取计划

2. 根据条件语句中谓词的选择度创建索引

因为建立索引需要占用数据库的存储空间，所以需要在空间和时间性能之间进行权衡。很多时候，只考虑那些在条件子句中有条件判断的列上建立的索引也会同样有效，同时节约了空间。例如例 5-15 中的查询，可以只建立(city，cntry_code)索引。还可以进一步地检查条件语句中这两个谓词的选择度：

```
查询:
1. select count(*) from temp.customer where city = 'WASHINGTON'
and cntry code = 'USA';
2. select count(*) from temp.customer where city = 'WASHINGTON';
3. select count(*) from temp.customer where cntry code = 'USA';
Results:
1. 1404
2. 1407
3. 128700
```

选择度越大，过滤掉的记录越多，返回的结果集也就越小。从上面的结果可以看到，第二个查询的选择度几乎和整个条件语句相同。因此可以直接建立单列索引(city)，性能与索引(city，cntry_code，add_date)相差不多。表 5-2 对这两个索引的性能和大小进行了对比。

表 5-2　两个索引的性能和大小对比

索　　引	查询总代价	索 引 大 小
cust_i1(city，cntry_code，add_date)	28.94 timerons	19.52MB
cust_i3(city)	63.29 timerons	5.48MB

从表 5-2 中可以看到，单列索引(city)具有更加有效的性能空间比，也就是说占有尽可能小的空间而得到尽可能高的查询速度。

3. 避免在建有索引的列上使用函数

这是一个很简单的原则，如果在建有索引的列上使用函数(函数的单调性不确定，函数的返回值和输入值可能不会一一对应)，就可能存在索引中位置差异很大的多个列值可以满足带有函数的谓词条件，DB2 优化器将无法进行 Matching Index Scan，更坏的情况下可能会导致直接进行表扫描。图 5-19 对比了使用函数前后存取计划的变化。

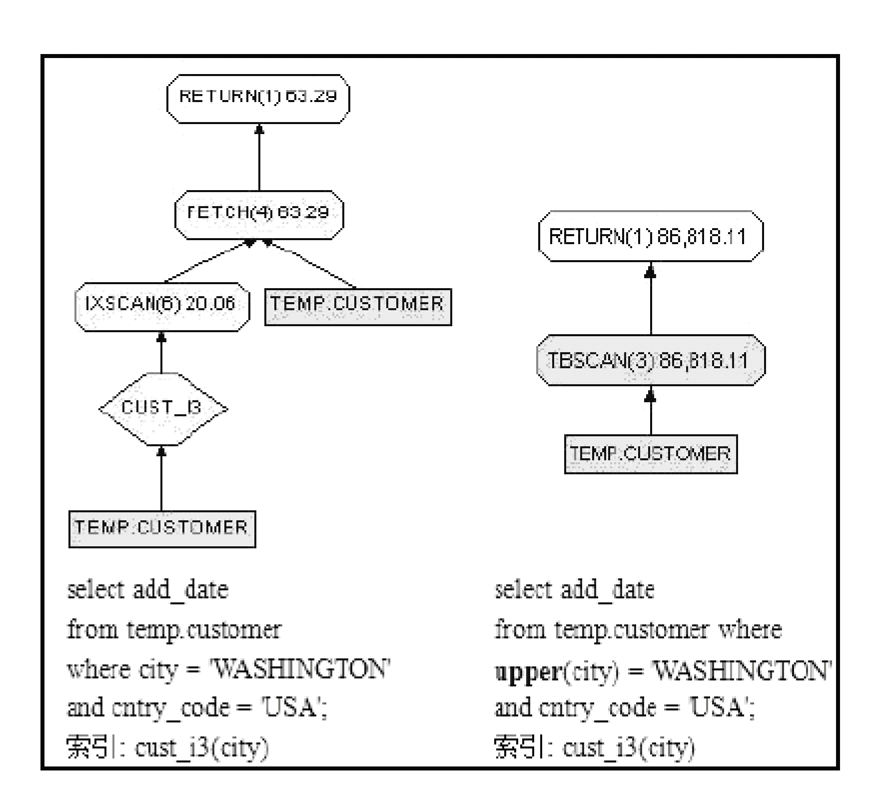


图 5-19 使用函数前后存取计划的变化

4. 在那些需要被排序的列上创建索引

这里的排序不仅仅指 ORDER BY 子句，还包括 DISTINCT、UNION 和 GROUP UP 子句，它们都会产生排序操作。由于索引本身是有序的，在创建过程中已经进行了排序处理，因此在应用这些语句的列上创建索引会降低排序操作的代价。这种情况一般针对没有条件语句的查询。如果存在条件语句，DB2 优化器会首先选择出满足条件的记录，然后才对中间结果集进行排序。对于没有条件语句的查询，排序操作在总的查询代价中会占有较大比重，因此能够较大限度地利用索引的排序结构进行查询优化。此时可以创建单列索引，如果需要创建复合索引，那么需要把被排序的列放在复合索引的第一列。图 5-20 对比了例 5-17 中的查询在创建索引前后的存取计划。

例 5-17 查询在创建索引前后的存取计划。

```
Select distinct add_date from temp.customer
```

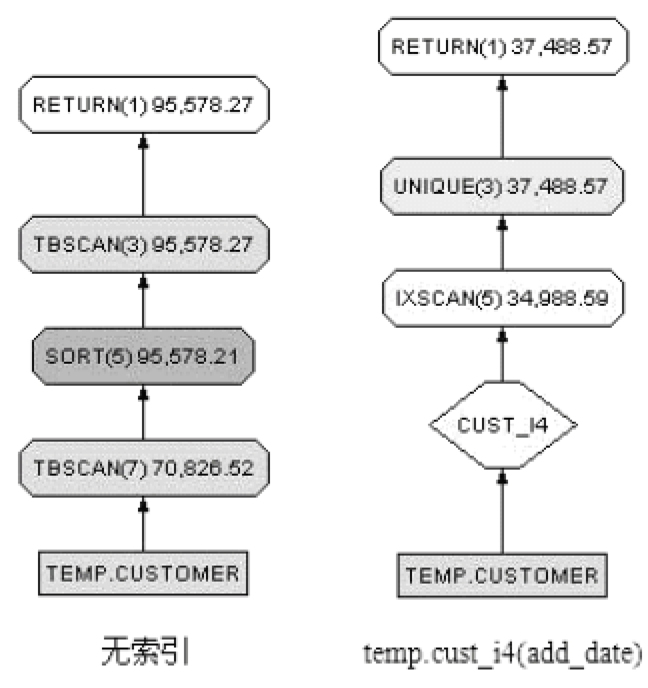


图 5-20 创建索引前后的存取计划

从图 5-20 中可以看到，在没有索引的情况下，SORT(排序)操作是 95578.21 timerons;但是在有索引的情况下，不再需要对结果集进行排序，可以直接进行 UNIQUE 操作，图中显示这一操作只花费了 37488.57 timerons.

图 5-21 对比了以下查询在创建复合索引前后的存取计划，从中可以更好地理解索引对排序操作的优化。

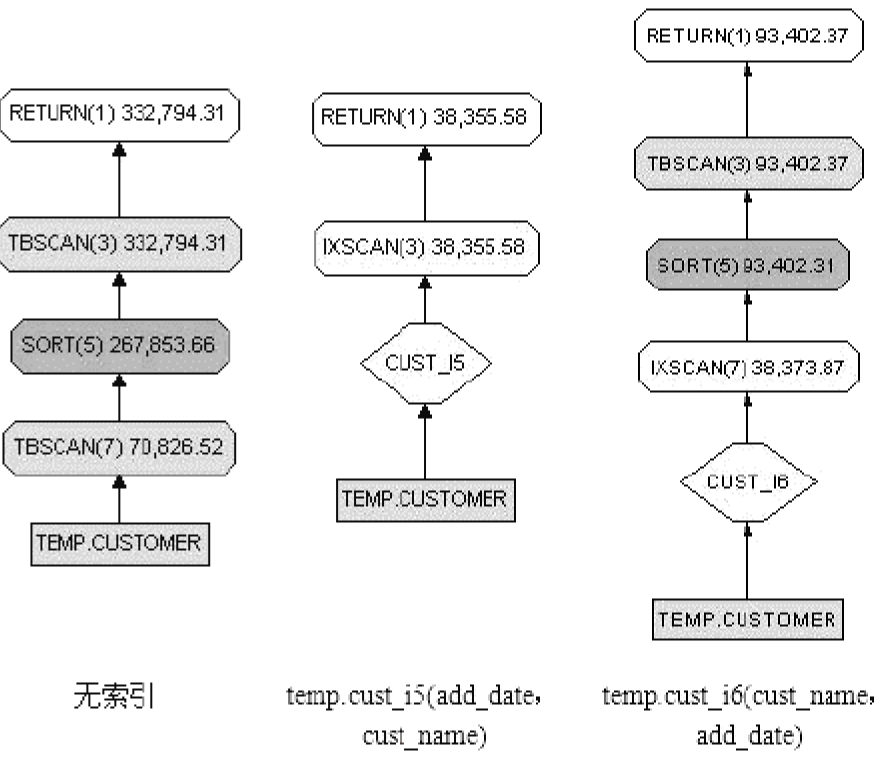


图 5-21　创建复合索引前后的存取计划

例 5-18　查询在创建复合索引前后的存取计划。

```
select cust_name from temp.customer order by add_date;
```

索引的 B+树结构决定了索引 temp.cust_i5 的所有叶子节点本身就是按照 add_date 排序的，所以对于上面的查询来说，只需要顺序扫描索引 temp.cust_i5 的所有叶子节点即可。但是对于 temp.cust_i6 索引，所有叶子节点都是按照 cust_name 排序的，因此在经过对索引的叶子节点扫描并获得所有数据之后，还需要对 add_date 进行排序操作。

5. 合理使用 include 关键词创建索引

对于类似下面的查询：

```
select cust name from temp.customer
where cust_num between '0007000000' and '0007200000'
```

在前面提到，可以在 cust_num 和 cust_name 上建立复合索引来提高查询性能。但是由于 cust_num 是主键，因此可以使用 include 关键字创建唯一索引：

```
create unique index temp.cust_i7 on temp.customer(cust_num) include(cust_name)
```

使用 include 后，cust_name 列的数据将只存在于索引树的叶子节点，而不存在于索引

的关键字中。这种情况下，使用带有 include 列的唯一索引会带来优于复合索引的性能，因为唯一索引能够避免一些不必要的操作，比如排序。对于上面的查询，创建索引 temp.cust_i7 后存取计划的代价为 12338.7 timerons，创建复合索引 temp.cust_i8(cust_num，cust_name) 后的代价为 12363.17 timerons。一般情况下，当查询的 WHERE 子句中存在主键的谓词时，就可以创建带有 include 列的唯一索引，形成纯索引访问来提高查询性能。注意 include 只能用在创建唯一索引中。

6. 指定索引的排序属性

下面是用来显示最近某个员工入职时间的查询：

```
select max(add_date) from temp.employee
```

很显然，这个查询会进行全表扫描。查询计划如图 5-22(a)所示。

显然可以在 add_date 上创建索引。根据下面命令创建索引后的查询计划如图 5-22(b)所示：

```
create index temp.employee_i1 on temp.employee(add_date)
```

这里存在一个误区，大家可能认为：既然查询里要取得的是 add_date 的最大值，而我们又在 add_date 上建立了索引，优化器应该知道从索引树中直接寻找最大值。但是实际情况并非如此，因为创建索引的时候并没有指定排序属性，默认为 ASC 升序排列，DB2 将会扫描整个索引树的叶子节点并取得所有值后，取其最大值。可以通过设置索引的排序属性来提高查询性能，根据下面命令创建索引后的查询计划如图 5-22(c)所示：

```
create index temp.employee_i1 on temp.employee(add_date desc)
```

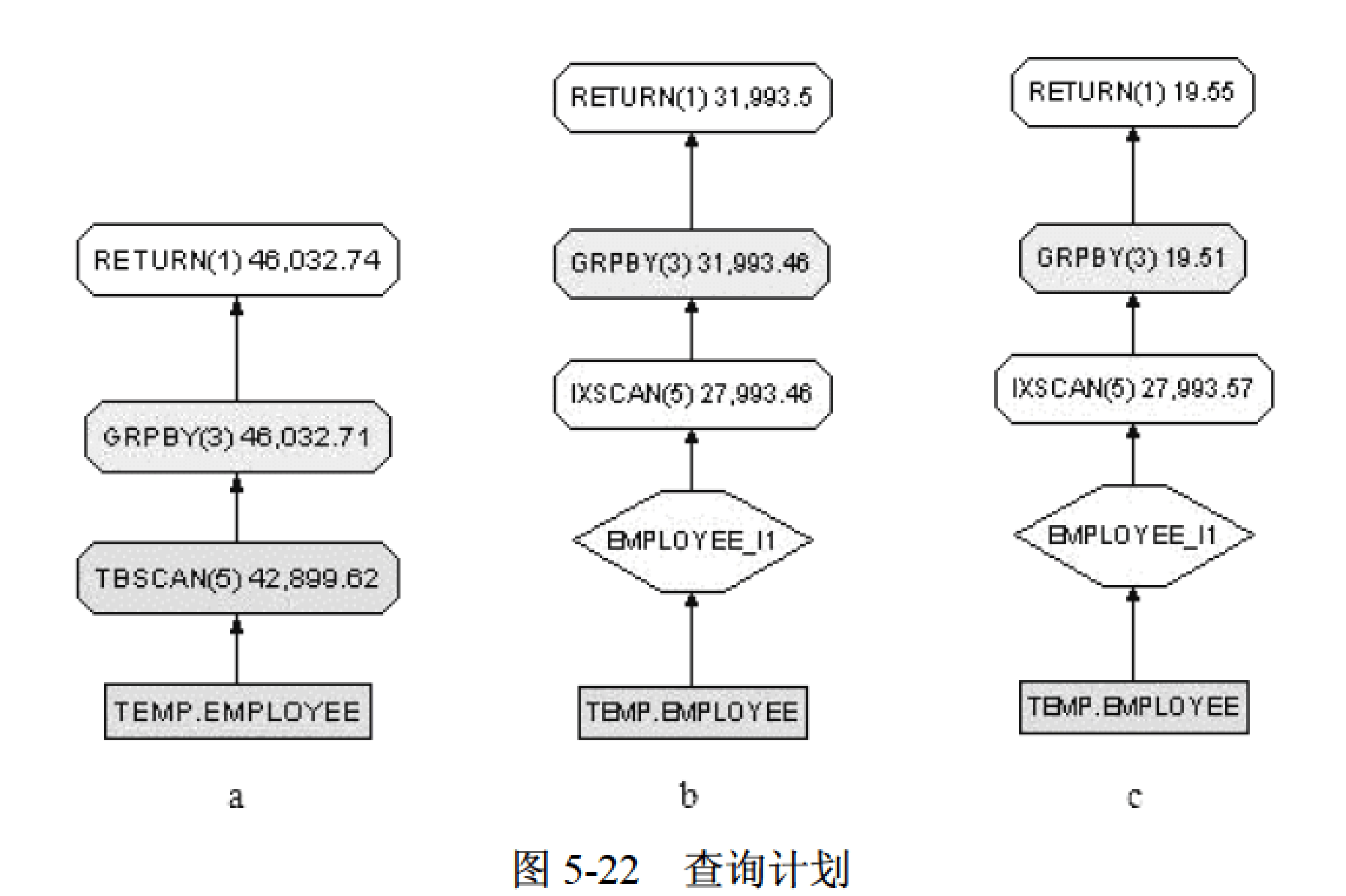


图 5-22　查询计划

对于降序排列的索引，DB2 不需要扫描整个索引数的叶子节点，因为第一个节点便是最大的。同样可以使用 ALLOW REVERSE SCANS 来指定索引为双向扫描，具有和 DESC 近似的查询性能。ALLOW REVERSE SCANS 可以被认为是 ASC 和 DESC 的组合，只是在以后更新数据的时候维护成本会相对高一些。

如果无法改变索引的排序属性，但又具有额外的信息，比如公司每个月都会有新员工入职，那么这个查询就可以改写成：

```
    select max(add date) from temp.employee where add date > current timestamp
-1 month
```

这样一来，通过限定查询范围也能有效地提高查询性能。

5.3.9　索引总结

创建的索引应该取决于数据和存取数据的查询。

以下准则可帮助你确定如何创建可用于各种目的的索引：

- 要避免某些排序，只要有可能，就使用 CREATE UNIQUE INDEX 语句定义主键和唯一键。
- 要改善数据检索，将 INCLUDE 列添加至唯一索引。合适的列为：
  ——被频繁存取，因此可从完全索引访问(index access only)受益的列。
  ——不需要用来限制索引扫描的范围的列。
  ——不影响索引键的排序或唯一性的列。
- 要有效存取小表，使用索引来优化对含有较多数据页的表的频繁查询，数据页数记录在 SYSCAT.TABLES 目录视图的 NPAGES 列中。应该：
  ——根据连接表时要使用的任何一列来创建索引。
  ——根据将用于定期索引特定值的任何列来创建索引。
- 要有效地搜索，对键使用升序还是降序取决于最常使用的次序。尽管当在 CREATE INDEX 语句中指定 ALLOW REVERSE SCANS 参数时可以按逆向方向搜索值，但是执行指定索引次序的扫描比执行逆向扫描稍微快一些。
- 要节省索引维护成本和空间：
  ——避免创建的索引是这些列上其他索引键的部分键。例如，如果列 a、b 和 c 上有索引，那么列 a 和 b 上的第二个索引一般用处不大。
  ——不在所有列上任意创建索引。不必要的索引不仅占用空间，而且会导致大量准备时间。当使用具有动态编程连接枚举的优化级别时，这对于复杂的查询特别重要。

——使用下列一般规则来确定将为表定义的索引的典型数目。此数目根据数据库的主要使用来确定：

对于在线事务处理(OLTP)环境，创建 2 个或 3 个索引。

对于只读查询环境，可以创建 5 个以上索引。

对于混合查询和在线事务处理环境，可以创建 2 到 5 个索引。

- 要改进对父表执行的删除和更新操作的性能，可以在外键上创建索引。
- 对于快速排序操作，在频繁用于数据排序的列上创建索引。
- 要改进多列索引的连接性能。如果第一个键列有多项选择，就使用最常用的“=”(等值连接)谓词指定的那一列，或使用像第一个键那样具有最多不同值的那些列。
- 要帮助新插入的行根据索引进行群集并避免页分割，定义集群索引。集群索引应显著减少重组表的需要。

  当定义表时使用 PCTREE 关键字来指定页上应该留下多少可用空间，才能允许将插入行适当地放在页上。也可以指定 LOAD 命令的 pagefreespace MODIFIED BY 子句。
- 要启用联机索引整理碎片，可以在创建索引时使用 MINPCTUSED 选项。MINPCTUSED 指定索引叶子页中最小使用空间量的阀值并启用联机索引整理碎片。如果这些删除实际上是从索引页除去键，那么可以在键删除期间以性能损失为代价而减少重组的需要。

在下列情况下，考虑创建索引：

- 在最频繁处理的查询和事务的 WHERE 子句中，在使用的那些列上创建索引。

  例如以下 WHERE 子句：

```
WHERE  WORKDEPT=A01 OR WORKDEPT=E21
```

  通常将会从 WOPKDEPT 上的索引获益，除非 WORKDEPT 列包含许多重复值。
- 在按查询需要的顺序对行排序的一列或多列上创建索引。不仅在 ORDER BY 子句中，而且其他功能，比如 DISTINCT 和 GROUP BY 子句也都需要排序。

  例如，以下示例使用 DISTINCT 子句：

```
SELECT DISTINCT WORKDEPT     FROM EMPLOYEE
```

  数据库管理器可以使用 WORKDEPT 上定义为升序或降序的索引来消除重复值。

  此时索引也可用于 GROUP BY 子句中，将值分组，如下所示：

```
SELECT WORKDEPT, AVERAGE(SALARY) FROM EMPLOYEE GROUP BY WORKDEPT
```

- 使用复合键创建索引，该键命名语句中引用的每个列。当用此方式指定索引时，可以从纯索引检索数据，这比存取表更有效。
  例如，考虑下列 SQL 语句：

```
SELECT LASTNAME  FROM EMPLOYEE  WHERE WORKDEPT IN('A00', 'D11', 'D21')
```

  如果为 EMPLOYEE 表的 WORKDEPT 和 LASTNAME 列定义索引，那么通过扫描索引而不是扫描整个表，可能会更有效地处理该语句。注意，因为谓词基于 WORKDEPT，因此该列应是索引的第一列。
- 使用 INCLUDE 列创建索引可改善表上索引的使用。使用上述示例，可将唯一索引定义为：

```
CREATE UNIQUE INDEX x ON employee(workdept) INCLUDE(lastname)
```

  指定 lastname 为 INCLUDE 列而不是索引键的一部分，意味着 lastame 只存储在索引的叶子页上。

5.4　使用序列提高性能

序列是数据库对象，允许自动生成值，例如支票号、流水号、订单号等。序列特别适合于生成唯一键值这一任务。应用程序可以使用序列来避免由用于跟踪数字的列值引起的可能的并行性和性能问题。与在数据库外部创建的数字相比，序列的优点在于数据库服务器会跟踪生成的数字。崩溃和重新启动不会导致生成重复的数字。

5.4.1　应用程序性能和序列

与其他方法相比，使用序列来生成值通常会提高应用程序的性能，这一点与标识列相同。序列的替代方法是创建存储当前值的单列表并使用触发器或在应用程序控制下递增值。但是，在分布式环境中，如果应用程序当前访问单列表，那么强制对表进行序列化访问所需的锁定可能会严重影响性能。

使用序列可以避免与单列表方法关联的锁定问题，并且可以将序列值高速缓存在内存中以减少响应时间。为了让使用序列的应用程序的性能最高，请确保序列高速缓存适当数量的序列值。CREATE SEQUENCE 和 ALTER SEQUENCE 语句的 CACHE 子句指定数据库管理器生成并存储在内存中的最大数目的序列值。

如果序列必须按顺序生成值，并且不会由于系统故障或数据库取消激活而在顺序中引入间隔，请在 CREATE SEQUENCE 语句中使用 ORDER 和 NO CACHE 子句。NO CACHE

子句保证生成的值中没有间隔，但会使应用程序性能降低一些，因为每次生成新值时，都会强制将序列写入数据库日志。请注意，由于事务回滚并且未真正使用它们请求的序列值，因此仍然会出现间隔。

生成的序列具有下列属性：

- 值可以是小数位为零的任何精确数字数据类型。这样的数据类型包括 SMALL INT、BIG INT、INTEGER 和 DECIMAL。
- 连续值之间可以有任何指定的整数增量。默认递增值是 1。
- 计数器值是可恢复的。当需要恢复时，从日志中重建计数器值。
- 可以高速缓存值以改善性能。在高速缓存中预分配并存储值，可以在为序列生成值时减少对日志的同步 I/O。在系统出现故障时，将认为尚未使用的所有高速缓存值已丢失。为 CACHE 指定的值是可能丢失的序列值的最大数目。

有两种表达式可与序列一起使用：

- NEXT VALUE 表达式：对指定序列返回下一个值。当使用 NEXT VALUE 表达式指定序列的名称时，将生成新的序号。但是，如果查询中有多个 NEXT VALUE 表达式的实例指定同一序列名，那么对于结果的每一行，序列计数器仅递增一次，并且 NEXT VALUE 的所有实例对结果的每一行返回同一个值。
- PREVIOUS VALUE 表达式：对于当前应用程序进程中的先前语句，该表达式对指定序列返回最新生成的值。也就是说，对于任何给定连接，PREVIOUS VALUE 将保持不变，即使另一个连接调用 NEXT VALUE 也是如此。

5.4.2 序列的设计原则

设计序列时，需要考虑标识列与序列之间的差别，以及哪个更适合你的环境。如果决定使用序列，那么需要熟悉可用的选项和参数。

可以通过调整序列的行为来满足应用程序要求。在发出 CREATE SEQUENCE 语句以创建新序列或对现有序列发出 ALTER SEQUENCE 语句时，可以更改序列的属性。除了容易设计和创建外，序列还具有其他各种选项，它们允许你更灵活地生成值：

- 从各种数据类型(SMALL INT、INTEGER、BIGINT 或 DECIMAL)中选择
- 更改起始值(START WITH)
- 更改序列增量，包括指定不断增大或不断减小的值(INCREMENT BY)
- 设置最小值和最大值，即序列的起始值和结束值(MINVALUE/MAXVALUE)
- 允许回绕值以便序列可以再次重新开始，或者禁止循环(CYCLE/NO CYCLE)
- 允许高速缓存序列值以提高性能，或者禁止高速缓存(CACHE/NO CACHE)

即使在生成序列后，这些值中的许多值也可以改变。例如，可能要根据星期几来设置

另一个起始值。使用序列的另一个实际示例是生成和处理银行支票。银行支票的序列号非常重要，如果一组序号丢失或损坏，那么将会产生严重后果。

为了提高性能，还应该了解并使用 CACHE 选项。此选项告知数据库管理器在系统生成多少个序列值后，才返回到目录以生成另一组序列。如果未指定 CACHE 值，那么默认值为 20。以默认值为例，在请求第一个序列值时，数据库管理器将自动在内存中生成 20 个连续值(1，2，……，20)。每次需要新的序号时，就会使用此内存高速缓存值来返回下一个值。用完此高速缓存值后，数据库管理器将生成下一组 20 个值(21，22，……，40)。

通过实现序列值的高速缓存，数据库管理器不必始终转至目录表来获取下一个值。这将减少与检索序列值相关的开销，但在系统出现故障或关闭时，可能还会导致序列中出现间隔。例如，如果决定将序列高速缓存设置为 100，那么数据库管理器将高速缓存 100 个这样的数字值，并且还将设置系统目录以表明下一个序列值应从 201 开始。在数据库关闭时，下一组序列值将从 201 开始。如果未使用生成的从 101 到 200 的数字，那么这些数字将从序列集中丢失。如果应用程序无法容忍生成的序列值中出现间隔，那么需要将高速缓存值设置为 NO CACHE，虽然这样会产生较高的系统开销。

注意：

只有在不需要唯一数字或者可以保证在序列循环后不再使用较旧的序列值时，才使用 CYCLE。

例如，如果要创建名为 xinzhuang_values 的序列，最小值为 0、最大值为 1000、NEXT VALUE 表达式使值递增 1，并且在达到最大值时返回到最小值，那么请发出以下语句：

```
CREATE SEQUENCE xinzhuang_values  START WITH 0
  INCREMENT BY 1 MAXVALUE 1000  CYCLE
```

5.4.3　维护序列

1. 创建序列

要创建序列，请使用 CREATE SEQUENCE 语句。与标识列属性不同，未使序列与特定表列相关，也未将之绑定至唯一表列，只是仅可通过该表列访问。

在可使用 NEXT VALUE 或 PREVIOUS VALUE 表达式的位置有几个限制。可以创建或改变序列，以便序列以下列其中一种方式生成值：

- 单调地递增或递减(按常量更改)且没有限制
- 单调地递增或递减至用户定义的限制并停止
- 单调地递增或递减至用户定义的限制并循环至起点，然后重新开始

注意：
在恢复使用序列的数据库时请务必小心。
对于在数据库外部使用的序列值(例如用于银行支票的序号)，如果将数据库恢复至数据库失败前的某个时间点，那么可能会导致对某些序列生成重复值。要避免可能的重复值，就不应该将在数据库外部使用序列值的数据库恢复至前一时间点。

例如，要使用所有选项的默认值创建名为 order_seq 的序列，在应用程序中或通过使用动态 SQL 语句发出以下语句：

```
CREATE SEQUENCE order_seq
```

此序列从 1 开始，并以 1 为增量增加且没有上限。

下面这条语句可以表示处理从 101 开始至 200 的一组银行支票。第一个顺序应该是从 1 到 100。序列从 101 开始并以 1 为增量增加，上限为 200。指定 NOCYCLE 以便不会产生重复的支票号。与 CACHE 参数关联的数指定了序列值的最大数目，数据库管理器预分配此数目并将之保存在内存中：

```
CREATE SEQUENCE order seq   START WITH 101 INCREMENT BY 1
     MAXVALUE 200     NOCYCLE     CACHE 25
```

生成顺序值

生成顺序值是常见的数据库应用程序开发问题。解决该问题的最好方法是在 SQL 中使用序列和序列表达式。每个序列是只能由序列表达式访问的唯一已命名数据库对象。

有两个序列表达式：PREVIOUS VALUE 和 NEXT VALUE。PREVIOUS VALUE 表达式对指定的序列返回应用程序进程中最新生成的值。与 PREVIOUS VALUE 表达式出现在同一语句中的任何 NEXT VALUE 表达式不会影响该语句中 PREVIOUS VALUE 表达式生成的值。NEXT VALUE 序列表达式使序列值递增并返回序列的新值。

要创建序列，请发出 CREATE SEQUENCE 语句。例如，要使用默认属性创建名为 xinzhuang_values 的序列，请发出以下语句：

```
CREATE SEQUENCE xinzhuang_values
```

要在序列的应用程序会话中生成第一个值，请使用 NEXT VALUE 表达式的 VALUES 语句：

```
VALUES NEXT VALUE FOR xinzhuang values
  1
-----------
```

```
    1
  1 record(s) selected.
```

要将序列值更新为序列的下一个值，请在 UPDATE 语句中包括 NEXT VALUE 表达式，如下所示：

```
UPDATE staff  SET id = NEXT VALUE FOR xinzhuang_values  WHERE id = 350
```

要使用序列的下一个值将新行插入到表中，请在 INSERT 语句中包括 NEXT VALUE 表达式，如下所示：

```
  INSERT INTO staff(id, name, dept, job) VALUES(NEXT VALUE FOR xinzhuang values,
'Kandil', 51, 'Mgr')
```

创建序列的几个示例

编写的许多应用程序需要使用序号来跟踪发票号、客户编号、订单号、流水号，以及每次需要新项时编号就会增大 1 的其他对象。通过使用标识列，数据库管理器可以使表中的值自动递增。虽然这项技术对于单独的表来说效果不错，但却可能不是生成多个表中需要使用的唯一值的最方便方法。

序列对象允许创建在程序员控制下递增并且可以在许多表中使用的值。以下示例说明了如何为客户编号创建数据类型为 INTEGER 的序号：

```
CREATE SEQUENCE customer_no AS INTEGER
```

默认情况下，序号从 1 开始并且每次递增 1，数据类型为 INTEGER。应用程序需要使用 NEXT VALUE 表达式来获取序列中的下一个值。此表达式生成序列的下一个值，然后可以将该值用于后续 SQL 语句：

```
VALUES NEXT VALUE FOR customer_no
```

程序员可以在 INSERT 语句中使用 VALUES 函数，而不是使用此函数生成下一个数字。例如，如果 Customer 表的第一列包含客户编号，那么可以按如下所示编写 INSERT 语句：

```
INSERT INTO customers VALUES  (NEXT VALUE FOR customer_no, 'comment', ...)
```

如果需要对插入到其他表中的操作使用序号，那么可以使用 PREVIOUS VALUE 表达式来检索先前生成的值。例如，如果需要将刚刚创建的客户编号用于后续发票记录，那么 SQL 应包括 PREVIOUS VALUE 表达式：

```
INSERT INTO invoices  (34,PREVIOUS VALUE FOR customer_no, 234.44, ...)
```

PREVIOUS VALUE 表达式可以在应用程序内多次使用，并且仅返回应用程序生成的最后一个值。后续事务可能已将序列递增至另一个值，但你看到的始终是生成的最后那个值。

2. 修改序列

使用 ALTER SEQUENCE 语句修改现有序列的属性。

可以修改的序列属性包括：

- 更改将来值之间的增量
- 建立新的最小值或最大值
- 更改高速缓存序号的数目
- 更改序列是否循环
- 更改是否必须按请求顺序生成序号
- 重新启动序列

有两种任务不是序列创建的一部分。它们是：

- RESTART：将序列复位为隐式或显式指定的值，该值是在创建序列时作为起始值指定的。
- RESTART WITH <numeric-constant>：将序列复位为准确的数字常数值。数字常数可以是任何正数值或负数值，而且任何小数点右边不带有非零数字。

在重新启动序列或更改为 CYCLE 之后，可能会生成重复序号。ALTER SEQUENCE 语句仅影响将来的序号。

不能更改序列的数据类型。而是必须删除当前序列，然后创建新序列，指定新的数据类型。

在改变序列时，会丢失数据库管理器未使用的所有高速缓存序列值。

3. 查看序列定义

使用包含 PREVIOUS VALUE 选项的 VALUES 语句来查看与序列相关的参考信息或查看序列本身。

要显示序列的当前值，请发出使用 PREVIOUS VALUE 表达式的 VALUES 语句：

```
VALUES PREVIOUS VALUE FOR xinzhuang values
          1
-----------
          1
          1record(s) selected.
```

可以重复检索序列的当前值，并且在发出 NEXT VALUE 表达式之前，序列返回的值不变。在以下示例中，PREVIOUS VALUE 表达式返回值为 1，直到当前连接中的 NEXT VALUE 表达式使序列的值递增为止：

```
VALUES PREVIOUS VALUE FOR xinzhuang_values
         1
 -----------
         1
         1record(s) selected.
VALUES NEXT VALUE FOR xinzhuang values
         1
   -----------
         2
         1record(s) selected.
```

即使另一个连接在同时使用序列值也是如此。

4. 删除序列

要删除序列，请使用 DROP 语句。
在删除序列时，语句的授权标识必须具有 SYSADM 或 DBADM 权限。
可以通过使用下列命令删除特定序列：

```
DROP SEQUENCE <sequence_name>
```

其中，<sequence_name>是要删除的序列名，包括隐式或显式模式名以正确标识现有的序列。不能使用 DROP SEQUENCE 语句删除系统为 IDENTITY 列创建的序列。一旦删除序列，就会删除对序列的所有特权。

5.4.4　比较序列与标识列

虽然对于 DB2 应用程序来说，序列和标识列用途相似，但它们之间存在如下重要差别：标识列使用 LOAD 实用程序自动生成单个表中的列值，序列根据请求使用 CREATE SEQUENCE 语句生成可在任何 SQL 语句中使用的顺序值。

1. 确定何时使用标识列或序列

虽然标识列和序列之间存在相似之处，但是也存在差别。在设计数据库和应用程序时可以使用各自的特征。
根据数据库设计和使用数据库的应用程序，下列特征将帮助你确定何时使用标识列以及何时使用序列。

标识列的特征

- 标识列自动为单个表生成值。
- 当将标识列定义为 GENERATED ALWAYS 时，始终由数据库管理器生成使用的值。在修改表的内容期间，不允许应用程序提供它们自己的值。
- 在插入行后，通过使用 IDENTITY_VAL_LOCAL()函数或 SELECT FROM INSERT 语句从插入中重新选择标识列，可以检索生成的标识值。
- LOAD 实用程序可以生成标识值。

序列的特征

- 未使序列与任何表相关。
- 序列生成可在任何 SQL 或 XQuery 语句中使用的顺序值。

由于任何应用程序都可以使用序列，因此有两种表达式可用来控制如何检索指定序列中的下一个值和正在执行的语句之前生成的值。对于当前会话中的先前语句，PREVIOUS VALUE 表达式对指定序列返回最新生成的值。NEXT VALUE 表达式对指定序列返回下一个值。使用这些表达式允许在几个表内的几条 SQL 和 XQuery 语句中使用相同值。

2. 标识列

允许数据库管理器自动为添加至表的每一行生成唯一数字值。如果正在创建表并且知道需要唯一标识将添加至该表的每一行，那么可通过 CREATE TABLE 语句向该表添加标识列：

```
CREATE TABLE <table name>
     (<column name 1> INT,
      <column name 2>, DOUBLE,
      <column name 3> INT NOT NULL GENERATED ALWAYS AS IDENTITY
       (START WITH <value 1>, INCREMENT BY <value 2>))
```

在上述示例中，第 3 列为标识列。可以定义的其中一个属性是，在添加行时用来唯一定义每一行的列中使用的值。INCREMENT BY 子句后面的值显示对于添加至该表的每一行来说，标识列内容的后续值的增量。

创建标识属性后，可以使用 ALTER TABLE 语句更改或除去这些属性。还可以使用 ALTER TABLE 语句在其他列中添加标识属性。

3. 序列

允许自动生成值。序列特别适合于生成唯一键值这一任务。应用程序可以使用序列，避免通过其他方法生成唯一计数器时引起的可能的并行性和性能问题。与标识列不同，未

使序列与特定表列相关，也未将之绑定至唯一表列，只是仅可通过该表列访问。

可以创建序列并在以后改变，以便通过无限递增或递减值来生成值；或者递增或递减至用户定义的限制，然后停止；或者递增或递减至用户定义的限制，然后循环至起点并重新开始。序列仅在单分区数据库中受支持。

以下显示了如何创建名为 orderseq 的序列：

```
CREATE SEQUENCE orderseq            START WITH1 INCREMENT BY 1
            NOMAXVALUE     NOCYCLE      CACHE 50
```

在上述示例中，序列从 1 开始，并以 1 为增量增加且没有上限。由于没有指定上限，因此没有理由循环至起点并从 1 重新开始。CACHE 参数指定了数据库管理器预分配并保存在内存中的序列值的最大数目。

5.5　视图

5.5.1　视图的类型

视图可从一个或多个表、昵称或视图中派生，并且可以在检索数据时与表互换使用。当对视图中显示的数据进行更改时，数据会在表中自行更改。在创建视图之前，视图基于的表、昵称或视图必须已经存在。

可以创建视图来限制对敏感数据的访问，同时又允许对其他数据进行更多的一般访问。

当插入到视图中，而视图定义中的选择列表直接或间接地包括表的标识列的名称时，标识列的同一规则也适用，就像 INSERT 语句直接引用表的标识列一样。

除按上述方式使用视图外，视图还可以用于：

- 改变表而不影响应用程序。这可通过创建基于基础表的视图来完成。使用基础表的应用程序不会因新视图的创建而受影响。新的应用程序可将创建的视图用于与那些使用基础表的应用程序不同的目的。
- 对一列中的值求和，选择最大值或计算平均值。
- 访问一个或多个数据源中的信息。可在 CREATE VIEW 语句内引用昵称，并可创建多个位置/全局视图(全局视图可以连接位于不同系统中多个数据源的信息)。当使用标准的 CREATE VIEW 语法创建引用昵称的视图时，将看到警告，警告目前视图用户的认证标识，而不是视图创建者的认证标识，用于访问数据源处的基本对象。使用 FEDERATED 关键字可以阻止此警告。

视图是高效率的数据呈现方法(无须维护数据)。视图不是实际的表，不需要永久存储器。“虚拟表”是即创建即使用的。

视图提供了另一种查看一个或多个表中数据的方法。视图和表一样具有列和行。可以像使用表一样将所有视图用于数据检索。是否可以在插入、更新或删除操作中使用视图取决于视图的定义。

视图可以包括自身所基于表中的所有或某些列或行。例如，可以在视图中连接部门表和职员表，以便可以列示特定部门中的所有职员。图 5-23 显示了表与视图之间的关系。

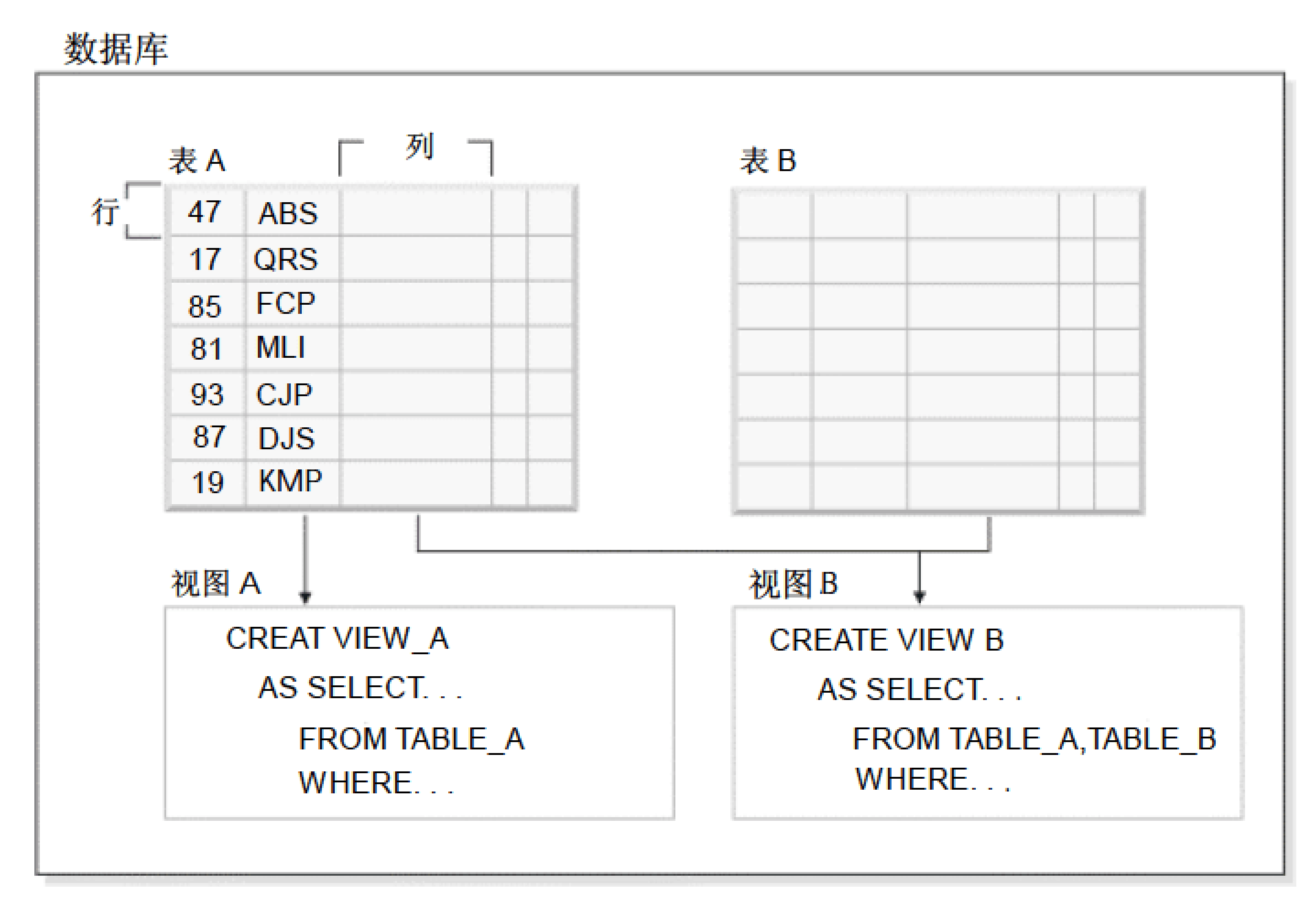


图 5-23　表与视图之间的关系

因为视图允许多个用户查看相同数据的不同表示，所以可以使用视图来控制对敏感数据的访问。例如，几个用户正在访问关于职员的数据表。经理可以看到关于他/她的职员的数据，但看不到关于其他部门中职员的数据；招聘专员可以看到所有职员的聘用日期，但看不到他们的薪水；财务人员可以看到薪水，但看不到聘用日期。这些用户中的每个用户都使用派生自表的视图。每个视图都显示为表，并且具有自己的名称。

当视图中的列直接派生自基本表的列时，视图中的列将继承适用于表列的所有约束。例如，如果视图包括其表的外键，那么使用视图的插入和更新操作时应遵守与表相同的引用约束。此外，如果视图的表是父表，那么使用视图的删除和更新操作时应遵守与对表执行删除和更新操作时相同的规则。

视图有以下几种类型：

系统目录视图

数据库管理器维护一组表和视图，这些表和视图包含关于数据库管理器所控制数据的信息。这些表和视图统称为系统目录。系统目录包含关于数据库对象(例如表、视图、索引、程序包和函数)的逻辑和物理结构信息，还包含统计信息。数据库管理器确保系统目录中的描述始终准确。

系统目录视图类似于任何其他数据库视图。可以使用 SQL 语句来查询系统目录视图中的数据。可以使用一组可更新的系统目录视图来修改系统目录中的某些值。关于系统目录视图的详细信息，请参见《高级进阶 DB2(第 2 版)》一书。

可删除视图

根据定义视图的方式，视图可以是可删除视图。可删除视图是可以对其成功发出 DELETE 语句的视图。

只有在遵循下列规则的情况下，视图才能被视为可删除视图：

- 外部全查询的每条 FROM 子句仅标识表(不带有 OUTER 子句)、可删除视图(不带有 OUTER 子句)、可删除的嵌套表表达式或可删除的公共表表达式。
- 数据库管理器应该能够使用视图定义来派生表中要删除的行。下列操作使视图变得不可删除：
  - ◊ 使用 GROUP BY 子句或列函数将多行分组为一行将导致原始行丢失并使得视图不可删除。
  - ◊ 同样，从 VALUES 派生行时，没有要删除行的表。视图也将不可删除。
- 外部全查询不使用 GROUP BY 或 HAVING 子句。
- 外部全查询的选择列表中不包括列函数。
- 外部全查询不使用集合操作(UNION、EXCEPT 或 INTERSECT)，但 UNION ALL 除外。
- UNION ALL 操作数中的表不能相同，并且每个操作数必须可删除。
- 外部全查询的选择列表不包括 DISTINCT。

视图必须符合上面列示的所有规则才能被视为可删除视图。例如，下列视图是可删除视图，因为遵循可删除视图的所有规则：

```
CREATE VIEW deletable view    (number, date, start, end)
    AS     SELECT number, date, start, end    FROM employee.summary
      WHERE date='01012007'
```

可插入视图

可插入视图允许使用视图定义来插入行。如果为视图定义了用于插入操作的 INSTEAD OF 触发器，或者视图中的至少一列可更新(与用于更新的 INSTEAD OF 触发器无关)，并且视图的全查询不包括 UNION ALL，那么视图是可插入视图。当且仅当给定行正好满足基础表的检查约束时，才能将行插入到视图中(包括 UNION ALL)。要插入到包含不可更新列的视图中，必须从列的列表中省略这些列。

下面创建的视图是可插入视图。但是在本例中，尝试插入视图将失败。这是因为表中存在不接受空值的列。这些列中的某些列未出现在视图定义中。尝试使用视图插入值时，数据库管理器会尝试将空值插入到 NOT NULL 列中，不允许执行此操作：

```
CREATE VIEW insertable_view    (number, name, quantity)
    AS   SELECT number, name, quantify FROM ace.supplies
```

可更新视图

可更新视图是一种特殊的可删除视图。如果可删除视图中的至少一列可更新，那么可删除视图就变成可更新视图。

当满足下列所有规则时，视图中的一列将可更新：

- 视图是可删除视图。
- 列解析为表列(不使用解引用操作)并且未指定 READ ONLY 选项。
- 如果视图的全查询包含 UNION ALL，那么 UNION ALL 操作数的所有相应列具有完全匹配的数据类型(包括长度/精度和小数位)以及完全匹配的默认值。

以下示例使用无法更新的常量值。但是，视图是可删除视图并且视图中的至少一列可更新。因此，视图是可更新视图：

```
    CREATE VIEW updatable view  (number, current date, current time,
temperature) AS SELECT number, CURRENT DATE, CURRENT TIME, temperature)
        FROM weather.forecast    WHERE number = 300
```

只读视图

如果视图不可删除、更新或插入，那么视图是只读视图。SYSCAT. VIEWS 目录视图中的 READONLY 列指示视图是只读(R)视图。

下面创建的视图不是可删除视图，因为使用了 DISTINCT 子句并且 SQL 语句涉及多个表：

```
    CREATE VIEW read only view   (name, phone, address)
     AS SELECT DISTINCT viewname, viewphone, viewaddress
```

```
      FROM employee.history adam, employer.deptSALES   WHERE adam.id = sales.id
```

5.5.2　创建 with check option 视图

下面显示了样本 CREATE VIEW 语句。基础表 EMPLOYEE 具有 SALARY 列和 COMM 列。为了安全起见，仅根据 ID、NAME、DEPT、JOB 和 HIREDATE 列创建此视图。此外，对 DEPT 列的访问受限制。此定义仅显示属于 DEPTNO 为 10 的部门的职员信息：

```
   CREATE VIEW EMP VIEW1     (EMPID, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
       AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE   WHERE DEPT=10;
```

在定义视图后，可以指定访问特权。由于只能访问表的受限视图，因此指定访问特权可以提供数据安全性。如上所示，视图可以包含 WHERE 子句以限制对某些行的访问，或者可以包含列的子集以限制对某些数据列的访问。

视图中的列名不必与基本表的列名匹配。表名具有关联的模式，视图名也一样。

定义视图后，就可以在诸如 SELECT、INSERT、UPDATE 和 DELETE 之类的语句中使用(但具有一些限制)。DBA 可以决定提供一组用户，他们对视图具有的特权级别比对表具有的特权级别要高。

创建带检查选项的视图

定义了 WITH CHECK OPTION 的视图将针对视图的 SELECT 语句强制检查任何修改或插入的行。使用检查选项的视图也称为对称视图。例如，仅返回部门 10 中职员的对称视图不允许插入其他部门的职员。因此，此选项将确保数据库中修改的数据的完整性，并在 INSERT 或 UPDATE 操作期间违反条件时返回错误。

如果应用程序无法将需要的规则定义为表检查约束，或者规则不适用于数据的所有用法，那么可以使用另一种方法在应用程序逻辑中实施规则。可以考虑创建表视图，表视图对数据的条件包括在指定的 WHERE 子句和 WITH CHECK OPTION 子句中。此视图定义将数据检索限于对应用程序有效的行集。此外，如果可以更新表视图，那么 WITH CHECK OPTION 子句将更新、插入和删除操作仅限于适用于应用程序的行。

不能对下列视图指定 WITH CHECK OPTION：

- 使用只读选项定义的视图(只读视图)。
- 引用 NODENUMBER 或 PARTITION 函数、非确定性函数(例如 RAND)或使用外部操作的函数的视图。

例 5-19　以下是使用 WITH CHECK OPTION 视图定义的示例，需要此选项以确保始终检查条件。该视图确保 DEPT 始终为 10，这将限制 DEPT 列的输入值。使用视图插入新值时，始终强制执行 WITH CHECK OPTION：

```
CREATE VIEW EMP_VIEW2(EMPNO, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
  AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE  WHERE DEPT=10
  WITH CHECK OPTION;
```

如果在 INSERT 语句中使用此视图，那么当 DEPTNO 列的值不是 10 时将拒绝行。一定要记住，在未指定 WITH CHECK OPTION 的情况下，在修改期间不会进行数据验证。

如果在 SELECT 语句中使用此视图，那么将会调用条件(WHERE 子句)并且生成的表仅包含匹配的数据行。也就是说，WITH CHECK OPTION 不影响 SELECT 语句的结果。

例 5-20 使用视图使程序和最终用户查询可以灵活地查看表数据。

下列 SQL 语句创建 EMPLOYEE 表的视图，用于列示部门 A00 的所有职员及其姓名和电话号码：

```
CREATE VIEW EMP_VIEW(DA00NAME, DA00NUM, PHONENO)
          AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
          WHERE WORKDEPT = 'A00'        WITH CHECK OPTION
```

此语句的第一行对视图命名并定义其中包含的列。名称 EMP_VIEW 在 SYSCAT.TABLES 中的模式内必须是唯一的。尽管不包含数据，视图名看上去仍像表名。该视图将拥有名为 DA00NAME、DA00NUM 和 PHONENO 的 3 列，它们与 EMPLOYEE 表中的列 LASTNAME、EMPNO 和 PHONENO 相对应。列示的列名按一一对应的关系应用于 SELECT 语句的选择列表。如果不指定列名，那么视图使用与 SELECT 语句的结果表的列相同的名称。

第二行是描述要从数据库选择哪些值的 SELECT 语句，可以包括子句 ALL、DISTINCT、FROM、WHERE、GROUP BY 和 HAVING。为视图提供列的数据对象的一个或多个名称必须跟在 FROM 子句后面。

5.5.3 维护视图

有些情况下，视图会变得不可用，不可用视图是指不能再用于 SQL 语句操作的视图。在下列情况下，视图可能变得不可用：

- 撤销了对基础表的特权。
- 删除了表、别名或函数。
- 删除它们从属的视图。

下列步骤可以帮助 DBA 恢复不可用视图：

(1) 确定最初用于创建视图的 SQL 语句。可以从 SYSCAT. VIEW 目录视图的 TEXT 列获取此信息。

(2) 将当前模式设置为 QUALIFIER 列的内容。

(3) 将函数路径设置为 FUNC_PATH 列的内容。

(4) 使用 CREATE VIEW 语句并使用相同的视图名和相同的定义来重新创建视图。

(5) 使用 GRANT 语句重新授予先前在视图上授予的所有特权(注意，在不可用视图上授予的所有特权都被撤销)。

如果不希望恢复不可用视图，可以使用 DROP VIEW 语句显式删除。例如，以下示例显示如何删除名为 EMP_VIEW 的视图：

```
DROP VIEW EMP_VIEW
```

也可以使用相同的名称和不同的定义来创建新视图。

5.6　表表达式

表表达式可用于代替定义视图，主要用在需要使用视图的结果表的查询语句中。视图的定义被存储在系统的编目表中，并且可由所有被授权的用户共享。表表达式是临时的，并且仅在包含它们的 SQL 语句生存期间有效，它们不能被共享，但比视图具有更多的灵活性，可以减少对系统目录表的维护，提高性能。

5.6.1　嵌套的表表达式

嵌套的表表达式(Nested Table Expression)可以被认为是视图，只是定义嵌套(被直接地定义)在查询语句的 FROM 子句中。

下面的查询语句使用嵌套的表表达式给出对如下询问的回答：对于教育级别高于 16 的那些雇员，总收入的平均值是多少？教育级别是什么？哪年雇佣的？

```
SELECT EDLEVEL, HIREYEAR, AVG(TOTAL PAY)
   FROM( SELECT EMPLNO,YEAR(HIREDATE) AS HIREYEAR,EDLEVEL,
                SALARY+BONUS+COMM AS TOTAL PAY FROM EMPLOYEE
  WHERE EDLEVEL >16 ) AS PAY LEVEL
    GROUP BY EDLEVEL,HIREYEAR
```

这个查询首先使用嵌套的表表达式从 HIREDATE 列中提取雇员的雇佣年号，以便在后边的 GROUP BY 语句中使用。在需要对表达式的结果进行分组时，表表达式是非常有用的。当打算进行类似的查询但使用不同的值作为对 EDLEVEL 的选择条件时，可能觉得利用表表达式更为方便，而不想创建为视图，因为创建视图需要增加更多的维护工作。

5.6.2　公用表表达式

公用表表达式(Common Table Expression)是被命名的结果表，定义在完全选择的前面，

并且以 WITH 关键字开始。赋给公用表表达式的标识符可在整个查询的任何 FROM 子句中用作表名。每次对公用表表达式名字的重复引用，使用的结果表都相同。这一点与嵌套的表表达式或视图是不同的，它们可能对每次引用给出具有不同结果的结果集。

下面的例子可完成如下要求：找出公司中这样的雇员，他们的教育级别高于 16，但收入低于与他们同时雇佣且具有相同级别的雇员收入的平均值。之后详细分析这个查询语句的各部分。

```
[1] WITH
   PAYLEVEL AS
     (SELECT EMPNO,YEAR(HIREDATE) AS HIREYEAR,EDLEVEL,
             SALARY+BONUS+COMM AS TOTAL_PAY
             FROM EMPLOYEE
             WHERE EDLEVEL>16
      ),
[2] PAYBYED(EDUC_LEVEL,YEAR_OF_HIRE,AVG_TOTAL_PAY) AS
       (SELECT EDLEVEL,HIREYEAR,AVG(TOTAL_PAY)
              FROM PAYLEVEL
              GROUP BY EDLEVEL,HIREYEAR
        )
[3] SELECT EMPNO,EDLEVEL,YEAR_OF_HIRE,TOTAL_PAY,AVG_TOTAL_PAY
     FROM PAYLEVEL,PAYBYED
       WHERE EDLEVEL=EDUC_LEVEL
              AND HIREYEAR=YEAR_OF_HIRE
              AND TOTAL_PAY<AVG_TOTAL_PAY
```

[1] 这是名为 PAYLEVEL 的公用表表达式，结果表含有雇员被雇佣的年号，雇员的总收入以及他们的教育级别，并且仅包含了教育级别高于 16 的雇员的信息。PAYLEVEL 使用的查询与前边例子中嵌套的表表达式相同。

[2] 这是名为 PAYBYED 的公用表表达式。PAYBYED(也即 PAY BY Education)使用在前边的公用表表达式中创建的 PAYLEVEL 表确定教育级别、雇佣年号，以及每种教育级别中雇员的平均收入，并且以不同于选择列表中采用的列名来命名这个表返回的列(比如 EDUC_LEVEL)。这个公用表表达式产生名为 PAYBYED 的结果集，与前边例子中嵌套的表表达式产生的结果集相同。

[3] 最后是给出希望的实际查询结果。两个表(PAYLEVEL 和 PAYBYED)被连接起来，以找出这样的雇员：他们的总收入低于与其在同一时间雇佣的职员的平均收入。注意，PAYLEVEL 在整个查询语句中使用了两次，而且在这两次查询的结果中，包含的行是相同的。

5.7　触发器设计

触发器定义一组操作，在响应对指定表的插入、更新或删除操作时将执行这些操作。执行这样的 SQL 操作时，触发器被认为是已激活的。触发器是可选的，并且可使用 CREATE TRIGGER 语句定义。

可将触发器与引用约束和检查约束配合使用，以强制执行数据完整性规则。还可使用触发器来完成更新其他表、自动生成或变换插入或更新的行的值，或者调用函数以执行诸如发出警报之类的任务。

对于定义或强制事务性业务规则而言，触发器是非常有用的机制，这些规则涉及数据的不同状态(例如，薪水增长不能超过 10%)。

使用触发器会设置逻辑以在数据库内强制使用业务规则，这表示应用程序不负责强制使用这些规则。对所有表强制使用的集中逻辑意味着更容易维护，因为在逻辑更改时，不需要更改应用程序。

使用触发器执行下列操作：

- 验证输入数据。
- 为新插入的行生成值。
- 为交叉引用而从其他表中进行读取。
- 为审计跟踪而向其他表写入。

可使用触发器支持一般形式的完整性或业务规则。例如，在接受订单或更新摘要数据表之前，触发器可以检查客户的信用额度。触发器具有以下优点：

- 更快地开发应用程序：因为触发器存储在数据库中，所以不必编写触发器在每个应用程序中执行的操作。
- 更容易维护：一旦定义触发器，那么当访问创建触发器所基于的表时，会自动调用触发器。
- 业务规则的全局实现：如果业务策略发生改变，只须更改触发器而不必更改每个应用程序。

5.7.1　触发器的类型

DB2 支持下列类型的触发器：

前触发器

在更新或插入操作前运行。在实际修改数据库之前，可以修改要更新或插入的值。可以将在更新或插入操作前运行的触发器用于下列几种用途：

- 在数据库中实际更新或插入值之前检查或修改这些值。如果需要将用户看到的数据格式变换为某种内部数据格式，这样做很有用。
- 运行用户定义的函数中编写的其他非数据库操作。

后触发器

在更新、插入或删除操作后运行。可以将在更新或插入操作后运行的触发器用于下列几种用途：

- 更新其他表中的数据。此功能对于保持数据之间的关系或保留审计跟踪信息很有用。
- 针对表或其他表中数据的检查。当引用完整性约束不适合或者表检查约束限制仅对当前表进行检查时，此功能对于确保数据完整性很有用。
- 运行用户定义的函数中编写的非数据库操作。在发出警报或更新数据库之外的信息时，此功能很有用。

INSTEAD OF 触发器

描述如何对视图执行插入、更新和删除操作，这些视图太复杂，以至于无法在本机支持这些操作。这种触发器允许应用程序将视图用作所有 SQL 操作(插入、删除、更新和选择)的唯一界面。

通常，INSTEAD OF 触发器包含视图主体中应用的逻辑的相反逻辑。例如，考虑用于解密源表中列的视图。此视图的 INSTEAD OF 触发器加密数据，然后将数据插入源表中，因此执行对称操作。

通过使用 INSTEAD OF 触发器，请求对视图执行的修改操作将替换为触发器逻辑，该逻辑代表视图执行操作。从应用程序的角度来看，这是透明地进行的，因为看到所有操作都是对视图执行的。只允许将单个 INSTEAD OF 触发器用于给定主题视图的每种操作。

视图本身必须是隐式类型视图或解析为隐式类型视图的别名。此外，不能是使用 WITH CHECK OPTION 定义的视图(对称视图)，或在其上直接或间接定义了对称视图的视图。

不同的触发器激活时间反映不同的触发器用途。基本上，前触发器是对数据库管理系统的约束子系统的扩展。因此，通常使用它们来：

- 验证输入数据。
- 自动生成新插入的行的值。
- 为交叉引用而从其他表中进行读取。

由于前触发器是在将触发器事件应用于数据库之前激活的，因此不使用它们来进一步修改数据库。因此，在检查完整性约束之前激活这些触发器。

相反，可以将后触发器视为每次特定事件发生时就在数据库中运行的应用程序逻辑的

模块。作为应用程序的一部分，后触发器始终看到处于一致状态的数据库。请注意，它们在完整性约束验证后运行。因此，主要将它们用来执行应用程序也可以执行的操作。例如：

- 在数据库中继续执行修改操作。
- 在数据库外执行操作，例如支持警报。请注意，回滚触发器时不会回滚在数据库外执行的操作。

比较而言，可以将 INSTEAD OF 触发器视为对定义该触发器的视图的反向操作的描述。例如，如果视图中的选择列表包含基于表的表达式，那么 INSTEAD OF INSERT 触发器的主体中的 INSERT 语句将包含反向表达式。

因为前触发器、后触发器和 INSTEAD OF 触发器具有不同的性质，所以可以使用一组不同的 SQL 操作来定义前触发器、后触发器和 INSTEAD OF 触发器的触发操作。例如，前触发器中不允许更新操作，这是因为不能保证触发操作不会违反完整性约束。同样，前触发器、后触发器和 INSTEAD OF 触发器支持不同的触发器粒度。

5.7.2　触发器创建示例

```
CREATE TRIGGER  1  trigger2
     2 AFTER  3 UPDATE OF  4 ON_HAND, MAX_STOCKED ON  5 PARTS
      6 REFERENCING NEW AS N OLD AS O
 7 FOR EACH ROW
 8 WHEN(N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
        9  BEGIN ATOMIC
      10  VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                                   N_ROW.ON_HAND,
                                   N_ROW.PARTNO));
        END@
```

1 触发器名称。

2 触发器触发的时间(BEFORE、AFTER 和 INSTEAD OF 子句)。

- 如果激活时间是 BEFORE，那么将在触发器事件执行之前对受影响的行集中的每行激活触发操作。因此，只有在前触发器完成每行的执行后才修改主题表。请注意，前触发器必须具有 FOR EACH ROW 粒度。
- 如果激活时间是 AFTER，那么将对受影响的行集中的每行或对语句激活触发操作，这取决于触发器粒度。此操作在触发器事件完成后，并且在数据库管理器检查触发器事件可能影响的所有约束(包括引用约束)后发生。请注意，后触发器可以具有 FOR EACH ROW 或 FOR EACH STATEMENT 粒度。

- 如果激活时间是 INSTEAD OF，那么将对受影响的行集中的每行激活触发操作，而不是执行触发器事件。INSTEAD OF 触发器必须具有 FOR EACH ROW 粒度，并且主题表必须是视图。其他触发器均无法将视图用作主题表。

3 触发器事件(INSERT、DELETE 或 UPDATE)。

4 触发器影响的列。

5 主题表。触发器建在主题表上的，是为主题表设计的业务逻辑。

6 使用转换变量访问触发器中的旧列值和新列值。

在行级触发器中，用 REFERENCING NEW AS N OLD AS O(称为伪记录)来访问数据变更前后的值。但要注意，INSERT 语句插入一条新记录，所以没有:old 记录。而 DELETE 语句删除一条已经存在的记录，所以没有:new 记录。UPDATE 语句既有:old 记录，也有:new 记录，分别代表修改前后的记录。引用具体的某一列值的方法是：

```
:old.字段名或:new.字段名
OLD AS correlation-name
```

指定相关名，捕获行的原始状态(在将触发操作应用于数据库之前)的中间结果集：

```
NEW AS correlation-name
```

指定相关名，捕获在将触发操作应用于数据库时，用于更新数据库中行的值的中间结果集。

7 触发粒度，有两种触发粒度：

```
FOR EACH ROW
```

运行的次数与受影响的行集中的行数相同。如果需要引用受触发操作影响的特定行，请使用 FOR EACH ROW 粒度：

```
FOR EACH STATEMENT
```

这会对整个触发器事件运行一次。如果受影响的行集为空(也就是说，当搜索式 UPDATE 或 DELETE 中的 WHERE 子句未限定任何行时)，那么 FOR EACH ROW 触发器不运行。但 FOR EACH STATEMENT 触发器仍运行一次。

8 定义触发器操作将触发的条件。

激活触发器将导致运行与触发器关联的触发操作。每个触发器正好有一个触发操作，而该触发操作又有两个组件：可选的触发操作条件或 WHEN 子句以及触发语句集。

触发操作条件是触发操作的可选子句，用于指定搜索条件，只有在搜索条件求值为 true 时才能运行触发操作内的语句。如果省略 WHEN 子句，那么始终执行触发操作内的语句。

9 受原子触发器影响的 SQL 语句要么全部成功，要么全部失败。
10 触发器的触发操作(触发语句)。

当创建原子触发器时，必须认真对待语句结束字符。默认情况下，命令行处理器将“;”当作语句结束标记。应该在脚本中手动编辑语句结束字符来创建原子触发器，以便使用非“;”字符。例如，可以用另一个特殊字符(如“#”)替换“;”，还可以在 CREATE TRIGGER DDL 的前面加上下列内容：

```
--#SET TERMINATOR @
```

要是更改了正在处理的 CLP 中的终止符，以下语法可用于复原：

```
--#SET TERMINATOR
```

要通过命令行创建触发器，请输入：

```
db2 -td@ -vf <script>
```

其中，delimitera 是备用语句的结束字符，而<script>是使用新<delimiter>的已修改脚本。

5.7.3 触发器设计总结

灵活地使用触发器可以方便实现业务逻辑，从而避免应用程序编码。但是在高并发的表上应尽量避免创建过多的触发器。

对于触发器，很多人认为不要使用，主要的原因是触发器不好控制并且会影响性能。在这里总结如下：触发器作为大型数据库的组成部分，可以代替应用程序编码来实现复杂的业务逻辑。有一些功能是其他方法无法代替的，特别是作为数据库约束的补充，所能进行的业务规则的约束，以及在跟踪和同步中所能起到的作用。

个人认为，触发器确实不好控制，这是因为：

- 触发器实现的是在表操作的同时，自动进行操作或控制，在写触发器代码时必须考虑其特殊性。
- 必须限定受触发器影响的记录，不能扩大。也就是说，必须引用转换变量，访问触发器中的旧列值和新列值：REFERENCING NEW AS N OLD AS O 中的两个虚表限定了操作的范围。
- 在 REFERENCING NEW AS N OLD AS O 中，由两个虚表限定的操作范围的作用域只限定触发器本身，调用的存储过程是不能使用的。
- 写触发器时必须考虑性能，因为其自动性。如果触发器性能不好，就有可能拖垮系统。

- 必须考虑一次操作多条记录的情况，除非保证一次只操作一条记录，一般不能用变量暂存虚表中的数据，否则就可能出现在批量操作情况下，触发器只处理最后一条记录的情况，这类错误可以说是触发器最常见的错误之一。
- 必须注意递归和嵌套触发器，因为触发器往往需要修改其他或本表数据来实现其功能，这里的修改数据往往能再次触发触发器，这时就必须保证嵌套或递归过程不是无限的，不会造成死循环，DB2 对触发器的嵌套层数有最多 16 层的限制。
- 触发器不好调试，比起一般的存储过程，触发器是在修改数据过程中触发，调试难度更大。调试过程必须考虑所有情况，比如空表插入数据、已有数据插入新数据、一次插入多行数据、修改一条数据、修改多条数据、一次删除多条数据、影响 0 行的修改或删除语句等等。

触发器不好控制，这就要求在决定是否使用触发器的时候非常谨慎。个人认为，对于约束功能，如果可以用其他数据库方法实现，比如唯一约束、外键约束、规则约束、不可空约束，就不要使用触发器，触发器只用来完成这些方法实现不了的约束。对于可以用触发器完成的跟踪、同步功能，请考虑是否必要，必要的时候才用。而对于特定业务需求实现的触发器，则需要与应用编程实现的优劣进行比较而做选择。

对于触发器影响性能的说法，笔者不是很同意，因为在触发器代码写得没有问题的前提下，触发器所做的工作并不是多余的，这些功能如果不在触发器中完成，就必须在存储过程中完成，或者在客户端用其他代码实现，因此在系统负担上基本是一样的。触发器唯一比其他实现方法多消耗的是触发器作为特殊的存储过程，必须有个被调用的过程。由于 DB2 有存储过程的预编译和过程缓存机制，因此这方面的开销不会很多。相反，在某些情况下，由于触发器是在记录更改过程中执行的，可以把某些服务器的负载分散到多个时间点去执行，因此一定程度上均衡了服务器负担，从而提高了整体性能。

5.8 例程

例程是封装程序和数据库应用逻辑的数据库对象。例程可以用来改进整体的数据库设计、数据库性能和数据安全性，以及实现基本审计机制等。例程可以是系统定义的，用户也可以使用 SQL 语句或编程语言来自己定义。不同类型的例程会提供不同的接口，这些接口可用来扩展 SQL 语句、客户机应用程序以及某些数据库对象的功能。

通过执行适合于例程类型的 CREATE 语句，在数据库中创建用户定义过程、函数以及方法。这些例程创建语句包括：

- CREATE PROCEDURE
- CREATE FUNCTION

- CREATE METHOD

特定于每个 CREATE 语句的子句定义例程的特征，例如例程名称、例程参数的数目和类型，以及例程逻辑的详细信息。DB2 使用这些子句提供的信息来标识何时调用例程，并在调用时运行例程。成功地执行用于创建例程的 CREATE 语句之后，就会在数据库中创建例程。例程的特征是存储在用户可以查询的 DB2 目录视图中。执行 CREATE 语句以创建例程的过程又称为定义例程或注册例程。

用户定义的例程定义存储在 SYSTOOLS 系统目录表模式中。下面是创建函数的例子：

```
CREATE FUNCTION updateInv(itemNo VARCHAR(20), amount INTEGER)
   RETURNS TABLE (productName VARCHAR(20),
                  quantity INTEGER)
   LANGUAGE SQL
   MODIFIES SQL DATA
   BEGIN ATOMIC

     UPDATE Inventory as I
       SET quantity = quantity + amount
         WHERE I.itemID = itemNo;

     RETURN
       SELECT I.itemName, I.quantity
         FROM Inventory as I
           WHERE I.itemID = itemNo;
   END
```

5.9　本章小结

本章向大家讲解了如何设计和创建表、索引、视图、触发器、例程等数据库对象。我们的关注点不应该放到创建命令上，而是应该放到每种对象的应用场合上。理解它们的特性并尽量用最合理的数据库技术来实现业务需求。其实还有一些概念本章没有讲解，例如，可以为表创建别名(alias)，在联合数据库中还可以为表创建昵称(nickname)，这些都不常用到。本章只是希望为大家讲解最常用的概念和最实用的技术。

第 6 章

数 据 移 动

在数据库的使用过程中，经常需要将一个数据库中的数据转移到另一个数据库中。我们常用的方法是利用某种类型的外部文件作为中介，将数据库中的某个(些)表中的数据导出到外部文件中，然后再把文件中的数据导入到另一个数据库中。

DB2 中实现以上功能的主要工具有 4 个：EXPORT、IMPORT、LOAD 和 db2move。其中，EXPORT 的功能是将表中的数据导出到外部文件中；而 IMPORT 和 LOAD 的功能是将外部文件中的数据导入到表中；db2move 工具可以将表导出到 IXF，然后使用 IMPORT CREATE 模式在不同的数据库中对其进行重建，从而对一组表在不同的系统间进行复制；IMPORT 和 LOAD 的功能类似，但在实现手段上有很大差异。本章主要讲解如下内容：

- 数据移动格式
- EXPORT
- IMPORT
- LOAD
- 数据移动注意事项
- db2move 和 db2look

6.1 数据移动格式

能够被 DB2 支持用作数据移动的中间文件的格式有 4 种：非定界 ASCII 文件(ASCII)、定界 ASCII 文件(DEL ASCII)、WSF 文件、PC/IXF 文件和游标，如图 6-1 所示。

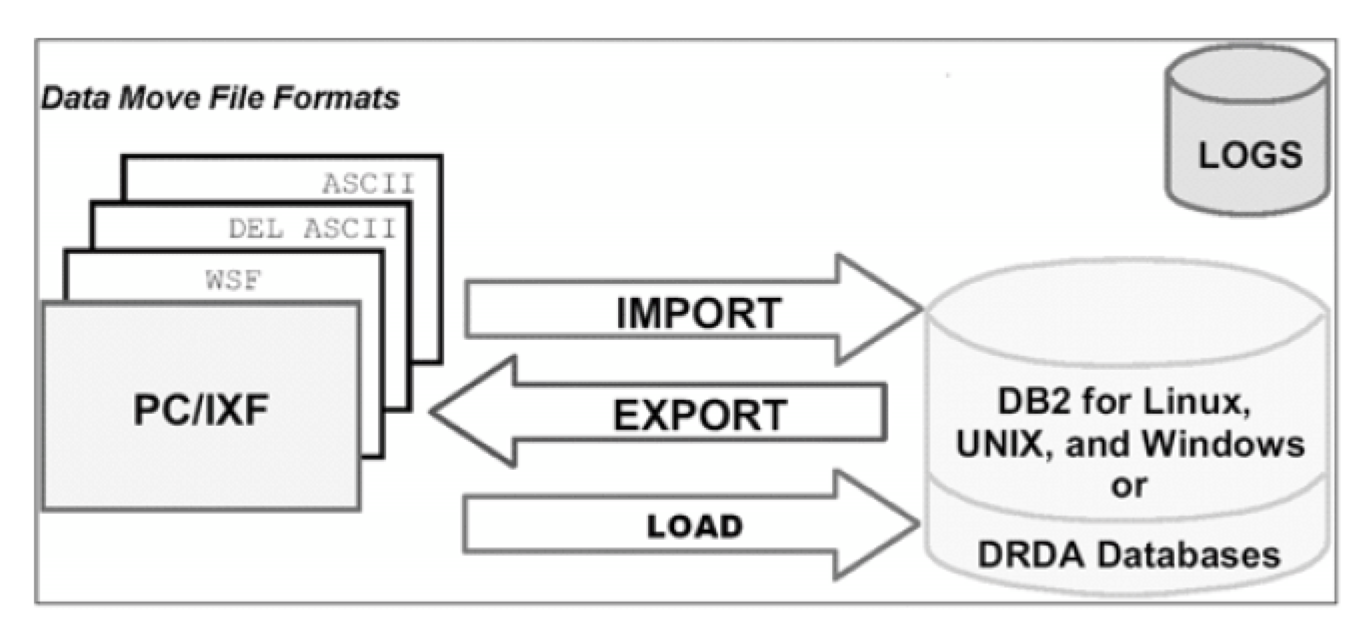


图 6-1　数据移动文件的格式

6.1.1　定界 ASCII 文件格式

定界 ASCII 文件是带有行定界符和列定界符的顺序 ASCII 文件。每个 DEL ASCII 文件都是一个 ASCII 字符流，ASCII 字符流由先按行排序然后按列排序的单元值组成。数据流中的行由行定界符分隔，行中的列值由列定界符分隔。文件类型修饰符可用于修改这些定界符的默认值。

DEL ASCII 文件示例

以下是 DEL ASCII 文件示例。每一行都以换行符序列结尾(在 Windows 操作系统中，每一行都以回车符/换行符序列结尾)：

```
"Smith, Bob",4973,15.46
"Jones, Bill",12345,16.34
"Williams, Sam",452,193.78
```

6.1.2　非定界 ASCII 文件格式

非定界 ASCII 文件也是 ASCII 字符流。数据流中的行由行定界符分隔，而行中的每一列则通过起始和结束位置来定义。

非定界 ASCII 文件格式(对于 IMPORT 和 LOAD 实用程序来说称为 ASC)有两种变体：定长 ASC 和变长 ASC。对于定长 ASC 来说，所有记录都是定长的。对于变长 ASC 来说，记录由行定界符(始终是换行符)定界。在非定界 ASCII 文件中，术语“非定界”表示列值未由定界符分隔。

在导入或装入 ASC 数据时，如果指定 reclen 文件类型修饰符，那么表示数据文件是定长 ASC 文件。如果未指定该修饰符，那么表示数据文件是变长 ASC 文件。

ASC 文件示例

以下是 ASC 文件示例。每一行都以换行符序列结尾(在 Windows 操作系统中，每一行都以回车符/换行符序列结尾)：

```
Smith, Bob           4973    15.46
Jones, Suzanne      12345    16.34
Williams, Sam      452123   193.78
```

6.1.3　PC/IXF 文件格式

PC/IXF 文件由一些可变长的记录组成，包括标题记录、表记录、表中每个列的列描述符记录，以及表中每行的一个或多个数据记录。PC/IXF 文件记录由一些包含字符数据的字段组成。PC/IXF 文件格式是一种通用关系数据库交换格式，支持很多关系数据类型，包括特定关系数据库产品可能不支持的某些类型。PC/IXF 文件格式保留了这一灵活性，例如，PC/IXF 体系结构同时支持单字节字符串(SBCS)和双字节字符串(DBCS)数据类型。PC/IXF 是一种非常通用的格式，被多种数据库管理系统所支持，可以用于在异种数据库间进行数据转移。

跨平台传输数据时，建议使用 PC/IXF 文件格式。PC/IXF 是在跨平台转换数据时推荐使用的文件格式，因为可以保留很多表属性，并且可以使用 DB2 数据移动工具以一种与平台无关的方式来处理数据。在 PC/IXF 文件中可以保留的表属性包括：主键、唯一约束、列信息(例如列名、数据类型和长度、可否为空属性以及标识属性)和索引信息。PC/IXF 文件中不能保留的表属性包括：引用和检查约束，以及部分列信息(例如默认值和生成的列属性)。

6.1.4　工作表文件格式

工作表文件格式(WSF)是一种专有的二进制文件格式，用于在 DB2、Lotus 1-2-3 和 Symphony 产品之间交换数据。WSF 文件不能被 LOAD 支持(要在导出操作期间创建与 WSF 格式相符的文件，可能会丢失一些数据)。如果在工作中不使用 Lotus 1-2-3 和 Symphony 产品，可以忽略这种数据格式。

6.1.5　游标

游标(cursor)是 SELECT 语句返回的结果集。这种格式仅限于 LOAD 使用，这种格式减少了中间文件的生成，可以提高 LOAD 的速度。

6.2 EXPORT

6.2.1 EXPORT 概述

下面我们首先来讲解 EXPORT 实用程序，它是几个实用程序中最简单的。EXPORT 实用程序会使用 SQL SELECT 语句或 XQuery 语句抽取数据，并将该信息放到文件中。EXPORT 工具的本质是把一条 SQL 语句的结果集导出到文件中。EXPORT 面向的是 SQL 而不单纯的是表。可使用输出文件移动数据以便将来执行导入或装入操作，或者将数据用于分析。

下列各项是基本导出操作所必需的：

- 用于存储已导出数据的操作系统文件的路径和名称。
- 输入文件中的数据格式：EXPORT 支持对输出文件使用 IXF、WSF 和 DEL 数据格式。
- 指定要导出的数据：对于大部分导出操作，需要提供 SELECT 语句指定需要进行检索以便导出的数据。

6.2.2 导出数据

使用 EXPORT 实用程序将数据从数据库导出至文件。该文件可使用若干外部文件格式中的一种。可以通过提供 SQL SELECT 语句来指定要导出的数据。

要想成功地调用 EXPORT 实用程序，必须拥有 SYSADM 或 DBADM 权限，或者拥有 EXPORT 命令中所访问的表或视图上的 CONTROL 或 SELECT 特权。在运行 EXPORT 实用程序之前，必须连接或能够隐式连接至要从中导出数据的数据库。因为实用程序会发出 COMMIT 语句，所以应在运行 EXPORT 实用程序之前发出 COMMIT 或 ROLLBACK 语句来完成所有事务并释放所有锁定。

EXPORT 相对简单且操作灵活的数据移动实用程序，可通过下列方法激活：通过控制中心、在 CLP 中发出 EXPORT 命令或调用 ADMIN_CMD 存储过程。

1. 使用控制中心导出数据

要使用“导出表”图形界面导出数据：

(1) 在“控制中心”中，展开对象树，直到找到“表”或“视图”文件夹为止。

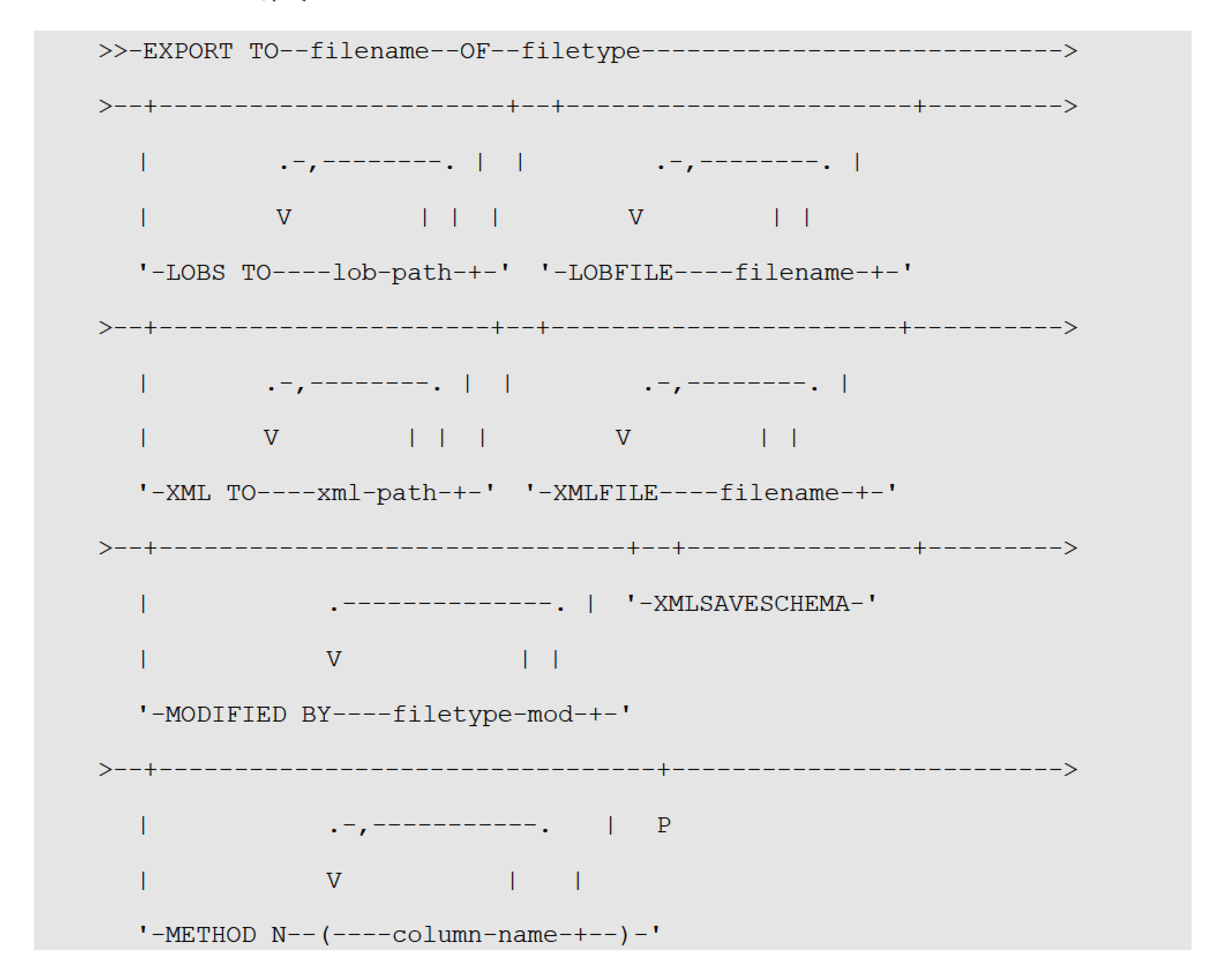
(2) 单击要使用的文件夹。任何现有的表或视图都将显示在窗口右边的窗格(内容窗格)中。

(3) 在内容窗格中，右击想要导出的表或视图，然后从弹出菜单中选择“导出”。“导出表”图形界面将打开，如图 6-2 所示，可以在界面中定制一些导出选项。



图 6-2　使用控制中心导出数据

2. EXPORT 命令



```
>>-EXPORT TO--filename--OF--filetype---------------------------->

>--+----------------------+--+----------------------+---------->

   |          .-,--------. |  |          .-,--------. |

   |          V          | |  |          V          | |

   '-LOBS TO----lob-path-+-'  '-LOBFILE----filename-+-'

>--+----------------------+--+----------------------+---------->

   |         .-,--------. |  |         .-,--------. |

   |         V          | |  |         V          | |

   '-XML TO----xml-path-+-'  '-XMLFILE----filename-+-'

>--+------------------------------+--+---------------+--------->

   |              .--------------. |  '-XMLSAVESCHEMA-'

   |              V              | |

   '-MODIFIED BY----filetype-mod-+-'

>--+--------------------------------+------------------------->

   |              .-,-----------.    |   P

   |              V             |    |

   '-METHOD N--(----column-name-+--)-'
```

```
>--+-----------------------+---------------------------------->
   '-MESSAGES--message-file-'
>--+-select-statement--------------------------------------+--><
```

export 常用命令参数如下：

- TO filename

指定导出文件的名称。

- OF filetype

指定导出文件的类型。EXPORT 支持的导出格式是 DEL、WSF 和 IXF 格式。

- LOBFILE filename

指定 LOB 文件的一个或多个基文件名。如果表中有 LOB 类型数据要被导出，就指定一个或多个文件，用于存储 LOB 数据。

导出操作中创建 LOB 文件时，通过从这个列表向当前路径(lob-path)追加当前基名，然后追加一个 3 位的序列号来构造文件名。例如，如果当前 LOB 路径为目录/u/foo/lob/path，并且当前 LOB 文件名为 bar，那么创建的 LOB 文件将为/u/foo/lob/path/bar.001、/u/foo/lob/path/bar.002 等等。

- LOBS TO lob-path

如果表中有 LOB 类型数据需要导出，就指定一个或多个目录文件，用于存储 LOB 数据。LOBFILE 参数指定的文件将存储在这个路径下。

- MESSAGES message-file

EXPORT 实用程序会将错误消息、警告消息和参考消息写至标准 ASCII 文本消息文件。对于 CLP 以外的所有接口，必须预先使用 MESSAGES 参数指定这些文件的名称。如果要使用 CLP 并且不指定消息文件，那么 EXPORT 实用程序会将消息写至标准输出。但是，您可能还想指定用于写入警告消息和错误消息的消息文件。为此，添加 MESSAGES 参数和消息文件名称。

- MODIFIED BY filetype-mod

指定一些额外的文件修饰符参数。文件类型修饰符提供了允许您更改数据、日期和时间戳记或代码页格式之类的许多选项，或者为输出文件指定特定的定界分隔符。

- METHOD N column-name

可指定要用于导出数据的不同列名。如果没有指定，将使用表中相应列的名字。导出表仅仅支持 N 方法。

- select-statement

利用 SELECT 语句指定要导出的数据。

- XMLFILE、XML TO 和 XMLSAVESCHEMA

可从包括一个或多个 XML 数据类型列的表中导出数据。使用 XMLFILE、XML TO 和 XMLSAVESCHEMA 参数指定有关如何存储已导出文档的详细信息。

我们来看看一个简单的导出数据的例子。下面的命令将 SELECT 语句的结果导出到一个 DEL 格式的文件中。消息文件 msg.out 用于记录有用的信息和遇到的错误或警告：

```
EXPORT TO myfile.del OF DEL   MESSAGES msg.out
SELECT staff.name, staff.dept, org.location   FROM org, staff
WHERE org.deptnumb = staff.dept;
```

其中，myfile.del 是要创建并导出的输出文件的名称，DEL 是文件格式，而 org 和 staff 是包含要导出的数据的表名。

3. 使用 ADMIN_CMD 存储过程

可以在命令行中直接调用 ADMIN_CMD 存储过程以导出数据，如下所示：

```
call sysproc.admin_cmd('export to /home/db2inst1/output/sales.del of del
messages /home/db2inst1/output/export.msg select * from sales')
```

6.2.3　导出数据示例

例 6-1　导出分界的文件。

```
EXPORT TO "D:\db2exp\employee.dat" OF DEL MESSAGES
"D:\db2exp\employee.log"      SELECT * FROM DB2ADMIN.EMPLOYEE;
```

例 6-2　将 XML 文档导出到单独的文件中。

在本例中，我们看看如何导出 XML 文档。以下命令导出 *XEmployee* 表。每个 XML 文档放在单独的文件中，这些文件的基本名称是 *XEmployee*。使用以下命令之后，XEmployee.del 包含文档的列表(比如<XDS FIL='XEmployee.001.xml' />)，而包含数据的实际文档(比如 XEmployee.001.xml)导出到 XML TO 选项指定的目录中。在本例中没有展示如何使用 XMLSAVESCHEMA 选项保存 XML 模式。

```
EXPORT TO "D:\db2XML\XEmployee.del" OF DEL XML TO "D:\db2XML\data"
XMLFILE "XEmployee" MODIFIED BY XMLINSEPFILES
MESSAGES "D:\db2XML\XEmployee.log"
SELECT * FROM "ALLAN WH THAM".XEMPLOYEE;
```

例 6-3　以下示例说明如何以 IXF 输出格式将 SAMPLE 数据库(用户必须连接至的数据库)的 STAFF 表中关于部门为 20 的职员的信息导出至 awards.ixf。

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff  where
dept = 20
```

例 6-4 以下示例说明如何将 LOB 导出到 DEL 文件：

```
db2 export to myfile.del of del lobs to mylobs/
 lobfile lobs1, lobs2 modified by lobsinfile     select * from emp_photo
```

例 6-5 以下示例说明如何将 LOB 导出到 DEL 文件，对可能无法装入到第一个目录中的文件指定第二个目录：

```
db2 export to myfile.del of del
 lobs to /db2exp1/, /db2exp2/ modified by lobsinfile   select * from emp_photo
```

例 6-6 以下示例说明如何将数据导出到 DEL 文件，将单引号用作字符串定界符，分号用作列定界符，逗号用作小数点。在将数据导回数据库时应使用同一约束：

```
Db2 export to myfile.del of del
   modified by chardel '' coldel; decpt,  select * from staff
```

6.3 IMPORT

6.3.1 IMPORT 概述

IMPORT 实用程序会使用 SQL INSERT 语句向表、类型表或视图填充数据。如果目标表和目标视图中已包含数据，那么输入数据可替换(replace)现有数据，也可追加(insert)至现有数据。

6.3.2 导入数据

IMPORT 实用程序将具有受支持文件格式的外部文件中的数据插入到表、层次结构、视图或昵称中。

为了使用 DB2 IMPORT，必须获得正确的权限和特权，否则就无法顺利地执行导入。表 6-1 列出了将文件导入数据库所需的权限和特权。

表 6-1 DB2 IMPORT 所需的权限和特权

操　作	权　限	特　权	注　释
创建新的表	SYDADM/DBADM	CREATETAB	DB2 IMPORT 允许在导入期间动态地创建新表。这只能应用于表

(续表)

操　　作	权　　限	特　　权	注　　释
在现有的表中插入数据	SYDADM/DBADM	CONTROL、INSERT 和 SELECT	可应用于表和视图
替换现有表中的数据	SYDADM/DBADM	CONTROL/(INSERT、SELECT、UPDATE 和 DELETE)	可应用于视图
在现有的表中追加数据	SYDADM/DBADM	SELECT 和 INSERT 特权	可应用于视图

在调用 IMPORT 实用程序前，必须连接至(或者能够隐式地连接至)要导入数据的数据库。由于实用程序将发出 COMMIT 或 ROLLBACK 语句，因此在调用导入之前，应该通过发出 COMMIT 或 ROLLBACK 操作以完成所有事务并释放所有锁定。

下列各项是基本导入操作所必需的：

- 输入文件的路径和名称
- 目标表或视图的名称或别名
- 输入文件中的数据格式
- 用于导入数据的方法

与 EXPORT 一样，IMPORT 是相对简单的数据移动实用程序。可通过下列方法激活：通过控制中心、发出 CLP 命令、调用 ADMIN_CMD 存储过程或通过用户应用程序。

1. 使用控制中心导入数据

(1) 在“控制中心”中，展开对象树，直到找到“表”文件夹为止。

(2) 单击“表”文件夹。所有现有表都会显示在窗口右边的窗格(内容窗格)中。

(3) 在内容窗格中右击想要导入的表，然后从弹出菜单中选择“导入”。这就打开了“导入表”图形界面，如图 6-3 所示，可以在图形界面中定制 IMPORT 选项。

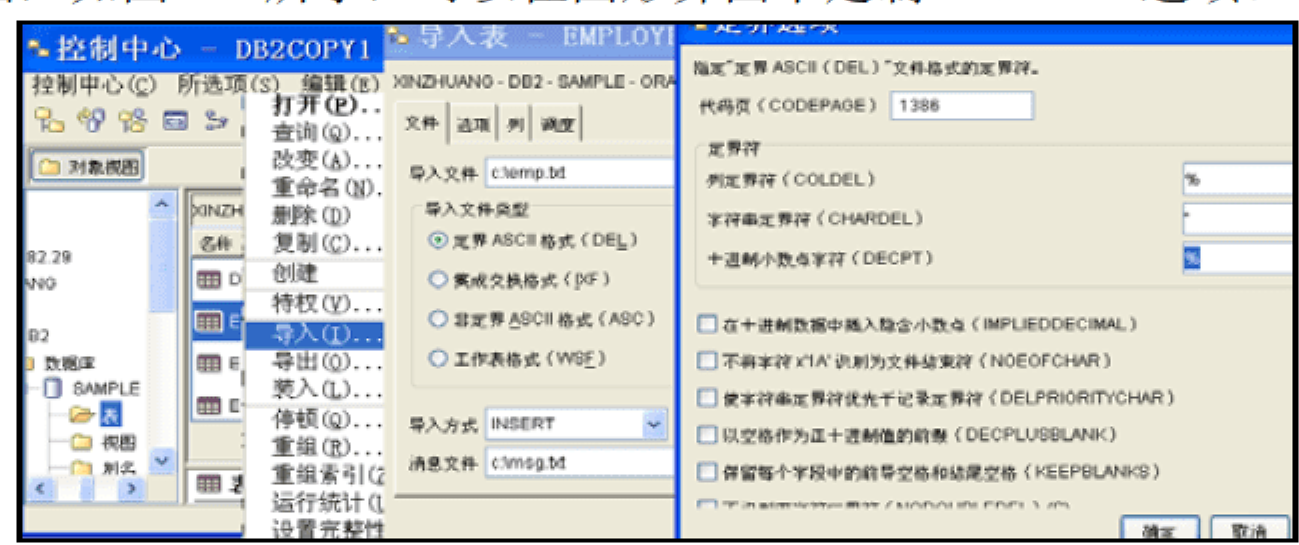


图 6-3　使用控制中心导入数据

2. IMPORT 命令

```
    >>-IMPORT FROM--filename--OF--filetype-------------------------->
    >--+-------------------------+--+-----------------------+------>
       |           .-,--------. |  |           .-,--------. |
       |           V          | |  |           V          | |
       '-LOBS FROM----lob-path-+-'  '-XML FROM----xml-path-+-'
    >--+-------------------------------+--------------------------->
       |             .--------------. |
       |             V              | |
       '-MODIFIED BY----filetype-mod-+-'
    >--+------------------------------------------------------------------
------------------------------------+-->
       |               .-,------------------------.
|
       |               V                          |
|
    '-METHOD--+-L--(----column-start--column-end-+--)--+-------------------
-----------------------------+-+-'
                |
|                  .-,-------------------.    | |
                |                                  |                  V
|    | |
                |                                  '-NULL
INDICATORS--(----null-indicator-list-+--)-' |
                |      .-,-----------.
|
                |      V            |
|

+-N--(----column-name-+--)---------------------------------------------
-----------------+
                |      .-,---------------.
|
                |      V                |
|

'-P--(----column-position-+--)-----------------------------------------
-----------------'
    >--+-----------------------------------+----------------------->
       '-XMLPARSE--+-STRIP----+--WHITESPACE-'
                   '-PRESERVE-'
    >--+------------------------------------------------------------------
```

```
-------------------+-->
       '-XMLVALIDATE USING--+-XDS--+-----------------------+--| Ignore and Map
parameters |-+-'
                            |      '-DEFAULT--schema-sqlid-'
|

+-SCHEMA--schema-sqlid------------------------------------------+
                            '-SCHEMALOCATION
HINTS-------------------------------------------'
       .-ALLOW NO ACCESS----.
    >--+--------------------+--+----------------------------+------->
       '-ALLOW WRITE ACCESS-'  '-COMMITCOUNT--+-n---------+-'
                                              '-AUTOMATIC-'
    >--+--------------------+--+-------------+--------------------->
       '-+-RESTARTCOUNT-+--n-'  '-ROWCOUNT--n-'
         '-SKIPCOUNT----'
```

IMPORT 常用命令参数如下：

- filename

指定进行数据导入的文件。

- OF filetype

指定导入文件的类型。导入上面提到的 4 种支持文件格式：DEL、WSF、IXF 和 ASC。

- LOBS FROM lob-path

如果表中有 LOB 类型数据需要导入，就指定一个或多个目录文件，用于存储 LOB 数据。

- MODIFIED BY filetype-mod

指定一些额外的文件类型修饰符参数。文件类型修饰符提供了允许您更改数据、日期和时间戳记或代码页格式之类的许多选项。

- 脱机导入(ALLOW NO ACCESS)

在 ALLOW NO ACCESS 方式下，导入会在插入任何行之前获取针对目标表的独占(X)锁定。挂起对该表的锁定有两种影响：

◊ 首先，如果其他应用程序挂起针对导入目标表的表锁定或行锁定，那么 IMPORT 实用程序将等待这些应用程序落实或回滚更改。

◊ 其次，IMPORT 实用程序运行时，请求锁定的任何其他应用程序会等待导入操作完成。

联机导入(ALLOW WRITE ACCESS)

- 在 ALLOW WRITE ACCESS 方式下，IMPORT 实用程序将获取针对目标表的非独占(IX)锁定。挂起对该表的此锁定具有下列影响：
  - ◊ 如果其他应用程序挂起不兼容的表锁定，那么在所有这些应用程序落实或回滚更改之前，IMPORT 实用程序不会开始插入数据。
  - ◊ IMPORT 实用程序运行时，如果任何其他应用程序请求不可兼容的表锁定，那么这些应用程序都将等待直至导入操作落实或回滚当前事务。注意，导入的表锁定仅对单个事务有效。因此，在每次提交后，联机导入必须请求表锁定并可能需要等待。
  - ◊ 如果其他应用程序挂起不兼容的行锁定，那么 IMPORT 实用程序将停止插入数据，直到所有这些应用程序落实或回滚更改。
  - ◊ IMPORT 实用程序运行时，如果任何其他应用程序请求不可兼容的行锁定，那么这些应用程序都将等待，直至导入操作落实或回滚当前事务。
  - ◊ 为保留联机属性并降低死锁几率，ALLOW WRITE ACCESS 导入将定期落实当前事务，并在上升为独占表锁定之前释放所有行锁定。如果未显式设置提交频率，那么导入会按指定 COMMITCOUNT AUTOMATIC 的方式落实。如果 COMMITCOUNT 设置为 0，那么不会执行任何落实。

- COMMITCOUNT n

每隔 n 条记录进行一次提交。避免在出现错误以后，需要重新导入所有的数据。比如，设定 n 为 100，那么系统每隔 100 条记录就进行一次提交，将导入的数据保存下来。如果在导入到 531 条记录时出现了错误，因为已经保留了前面的 500 条记录，只需要从第 501 条记录开始导入即可。也可以设置为 AUTOMATIC，让 DB2 自动设置 COMMITCOUNT。

- RESTARTCOUNT n

一般用于在导入过程失败了之后，定义重新进行导入的起点。对于前面的例子而言，n 应该设定为 501。

- MESSAGES message-file

用于指定在导入表过程中生成的警告信息和错误信息的存储文件，如果没有指定，将导出信息到标准导出上，如屏幕。一般来说，在导入完成后，应该检查这个文件中的信息。其中，比较重要的信息包括要导入的数据行的数目、成功导入的行的数目，以及被拒绝导入的行的数目。

导入方式

导入可使用 5 种方式，它们用于确定导入数据的方法。前 3 种方式为 INSERT、INSERT_UPDATE 和 REPLACE，在目标表已存在的情况下使用，如表 6-2 所示。这 3 种方式都支持 IXF、WSF、ASC 和 DEL 数据格式。但是，只有 INSERT 和 INSERT_UPDATE 可与昵称(nickname)配合使用。

表 6-2 导入数据的前 3 种方式

方　式	工 作 机 制
INSERT	将输入数据插入到目标表中而不更改现有数据
INSERT_UPDATE	将导入的数据行插入到表中，如果导入的数据与表中原来的数据主键一样，就执行更新(update)操作，否则执行插入(insert)操作。表中有主键时，才可以使用这种模式
REPLACE	删除所有现有数据并插入已导入数据，同时保留表和索引定义

另外两种方式为 REPLACE_CREATE 和 CREATE，在目标表不存在时使用，如表 6-3 所示。它们只能与 PC/IXF 格式的输入文件配合使用，此格式包含要创建的表的结构化描述。如果对象表具有自身以外的任何从属，那么不能以这些方式执行导入。

注意：

不建议使用导入的 CREATE 和 REPLACE_CREATE 方式。请改用 db2look 实用程序。

表 6-3 导入数据的后两种方式

方　式	最佳实践用法
REPLACE_CREATE	删除所有现有数据并插入已导入数据，同时保留表和索引定义。如果目标表和索引不存在，那么创建目标表和索引
CREATE	创建目标表和索引，可指定在其中创建新表的表空间名称

下面是一个使用 CLP 导入数据的例子：

```
db2 import from employee.txt of del messages msg.out  insert into employee
```

3. 使用 ADMIN_CMD 存储过程

可以在命令行中直接调用 ADMIN_CMD 存储过程以导出数据，如下所示：

```
call sysproc.admin_cmd('import from /home/db2inst1/output/sales.del of del
messages /home/db2inst1/output/export.msg insert into sales')
```

导入的工作机制

导入所需的步骤数和时间量取决于要移动的数据量和指定的选项。导入操作遵循下列步骤：

(1) 锁定表

根据您是否允许对表进行并行访问，导入会获取对现有目标表的独占(X)或非独占(IX)锁定(关于锁这部分信息，大家先简单了解)。

IMPORT 实用程序支持两种表锁定方式：脱机或 ALLOW NO ACCESS 方式；以及联机或 ALLOW WRITE ACCESS 方式。ALLOW NO ACCESS 方式会阻止并行应用程序访问表数据，ALLOW WRITE ACCESS 方式允许并行应用程序同时对导入目标表进行读写访问。如果未显式指定任何方式，那么导入会以默认方式 ALLOW NO ACCESS 运行。同时，在默认情况下，IMPORT 实用程序会使用隔离级别 RS(读稳定性)绑定至数据库。

(2) 查找和检索数据

导入使用 FROM 子句来查找输入数据。如果命令指示 XML 或 LOB 数据存在，那么导入会查找此数据。

(3) 插入数据

导入会替换现有数据或将新的数据行添加至表。

(4) 检查约束和激发触发器

写入数据后，导入会确保每个已插入行符合针对目标表定义的约束。有关被拒绝行的信息将写至消息文件。导入还会激发现有触发器。

(5) 提交操作

导入会保存所做更改并释放针对目标表的锁定，还可指定在导入期间定期落实。

6.3.3 导入示例

例 6-7 使用命令 CLP 导入 XML 文档。

```
IMPORT FROM "C:\XML\data\import.del"
OF DEL XML FROM "D:\XML\data" METHOD P (1)
   MESSAGES "C:\XML\xmlemp1.log"       INSERT INTO DB2ADMIN.XMLEMP (EMP);
```

其中的 D:\XML\data\import.del 包含指向实际文档的行指针。import.del 示例文件的内容如下：

```
 "<XDS FIL='emp.001.xml' />"
     "<XDS FIL='emp.002.xml' />"
```

```
    "<XDS FIL='emp.003.xml' />"
    "<XDS FIL='emp.004.xml' />"
    "<XDS FIL='emp.005.xml' />"
    "<XDS FIL='emp.006.xml' />"
    "<XDS FIL='emp.007.xml' />"
```

在成功装载之后，应该会在消息文件中看到下面这样的消息：

```
SQL3109N  The utility is beginning to load data from file
"D:\XMLPoT\labdoc\scripts\data\import.del".
SQL3110N  The utility has completed processing.
"42" rows were read from the input file.
SQL3221W  ...Begin COMMIT WORK. Input Record Count = "42".
SQL3222W  ...COMMIT of any database changes was successful.
SQL3149N  "42" rows were processed from the input file.
"42" rows were successfully inserted into the table.
"0" rows were rejected.
```

例 6-8　以下示例显示导入以管道符分界的文件：

```
IMPORT FROM "D:\db2out\employee.txt" OF DEL MODIFIED BY COLDEL|
MESSAGES "D:\db2out\employee2.log"      INSERT INTO DB2ADMIN.EMPLOYEE
```

IMPORT 实用程序存在下列限制：

- 如果现有表是父表，并且其中包含的主键被从属表中的外键引用，那么不能替换此表的数据，而只能追加数据。
- 不能执行导入替换操作来将数据导入到以立即刷新方式定义的物化查询表所关联的基表中。
- 不能将数据导入到系统表、摘要表或其他带有结构化类型列的表中。
- 不能将数据导入到已声明临时表中。
- 不能通过 IMPORT 实用程序创建视图。
- 根据 PC/IXF 文件创建表时，不会保留引用约束和外键定义。如果数据先前是使用 SELECT * 导出的，那么保留主键定义。
- 由于 IMPORT 实用程序会生成自己的 SQL 语句，因此在某些情况下可能会超过最大语句大小(2MB)。
- 不能使用 CREATE 或 REPLACE_CREATE 导入选项重新创建分区表或多维集群表(MDC)。
- 不能重新创建包含 XML 列的表。
- 不能导入已加密的数据。

- 导入替换操作不能识别 NLI(Not Logged Initially)子句。IMPORT 命令的 REPLACE 选项不能识别 CREATE TABLE 语句的 NOT LOGGED INITIALLY 子句或 ALTER TABLE 语句的 ACTIVATE NOT LOGGED INITIALLY 子句。如果包含 REPLACE 操作的导入操作与调用了 NLI 子句的 CREATE TABLE 或 ALTER TABLE 语句在同一事务内执行，那么此导入操作不能识别 NLI 子句。将记录所有插入操作。可以使用以下两种变通方法：
  - ◊ 使用 DELETE 语句删除表的内容，然后使用 INSERT 语句调用导入操作。
  - ◊ DROP 后重新创建该表，接着使用 INSERT 语句调用导入操作。

6.4 LOAD

6.4.1 LOAD 概述

我们在前面已经讲解了 IMPORT 实用程序，IMPORT 本质上是执行 INSERT、UPDATE 和 DELETE SQL 语句。所以在把数据放到表中时，会触发触发器、执行日志记录并执行约束检查和索引构建。由于 LOAD 实用程序直接将格式化的数据页(data page)写入数据库，这会绕过触发器和日志记录机制，因此 LOAD 实用程序不会触发触发器，并且除了验证索引唯一性以外不执行引用约束检查或表约束检查。LOAD 实用程序能够高效地将大量数据移到新创建的表或者已包含数据的表中。此实用程序能够处理绝大多数数据类型，其中包括 XML、大对象(LOB)和用户定义的类型(UDT)。下面我们介绍 LOAD。

LOAD 过程由 4 个不同的阶段组成，如图 6-4 所示。

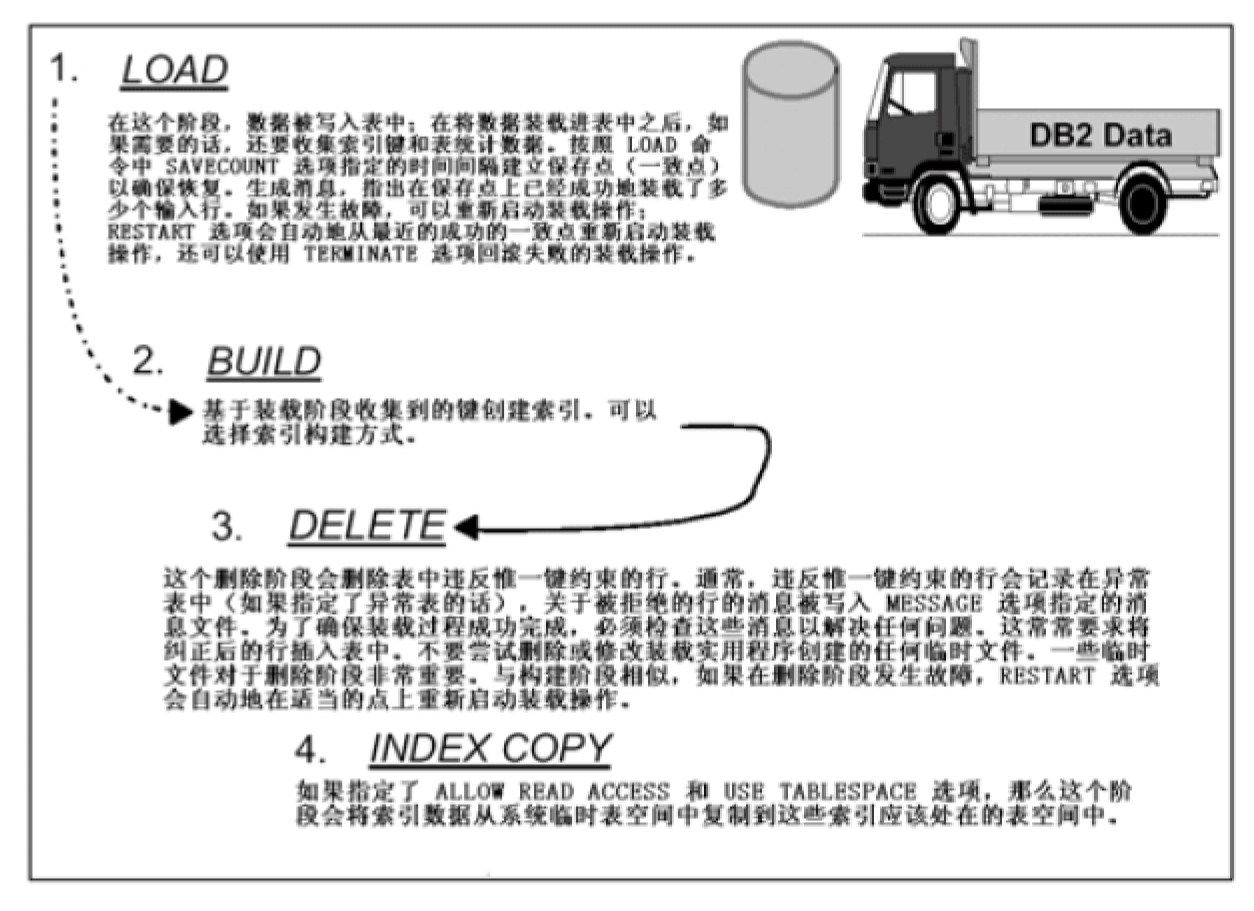


图 6-4　LOAD 过程

6.4.2　装入数据

LOAD 实用程序能够高效地将大量数据移到新创建的表或者已包含数据的表中。在调用 LOAD 实用程序前，您必须连接至(或者能够隐式地连接至)要装入数据的数据库。由于该实用程序将发出 COMMIT 语句，因此应该通过发出 COMMIT 或 ROLLBACK 语句来完成所有事务并释放所有锁定，然后再调用 LOAD 实用程序。

与 DB2 IMPORT 相似，DB2 LOAD 要求授予某些权限和特权。需要 SYSADM 或 DBADM 权限，至少要有 LOAD 权限和相关的 INSERT 或 DELETE 特权。

可以通过命令行处理器(CLP)、控制中心中的“装入”向导或者调用 ADMIN_CMD 存储过程来调用 LOAD 实用程序。

1. 使用控制中心

尽管命令行方式灵活而且强大，但是在命令提示语法中选择众多的选项是很麻烦的。调用 LOAD 实用程序的一个简便方法是使用“控制中心”，它会以向导驱动的方式为用户提供在线帮助。“控制中心”可以引导用户轻松地实现成功的装载，即使用户不熟悉 LOAD 也没关系。在下面的步骤中，将一个分界的文件装载进表 *employee* 中，从而体会一下如何在“控制中心”中进行数据装载。

(1) 选择表 *employee* 并选择“装入”，从而调用 LOAD 实用程序。这时会显示图 6-5 所示的对话框。

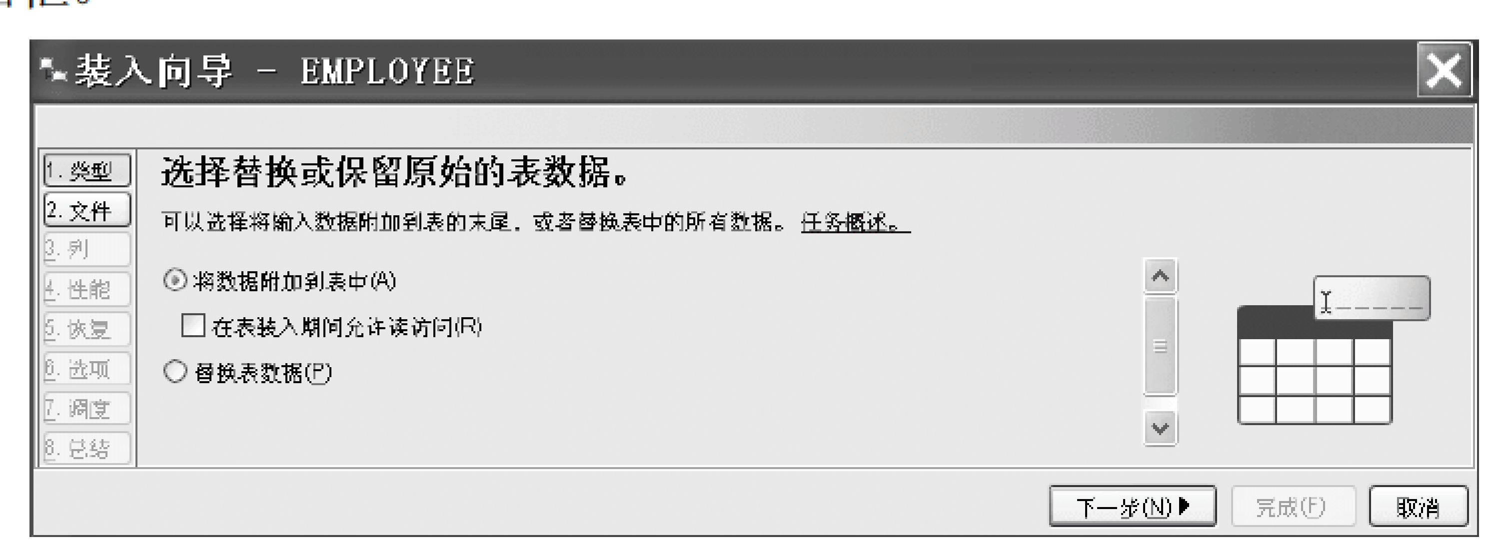


图 6-5　选择追加还是替换

注意，这个向导有 8 个步骤。但是，可以直接单击“下一步”选用默认设置。每个步骤都提供了不同的选项，根据您的装载需求，其中一些选项是必需的。在这个简单的演示中，第一步采用默认的“将数据附加到表中”。在这种模式下，允许用户在装载期间访问数据。单击“下一步”继续。

(2) 第二步带领用户选择文件格式(默认格式是 DEL)。在这一步中，选择输入文件和消息文件的位置——本地(调用 LOAD 实用程序的地方)还是远程。还可以指定要处理的总

行数。单击“下一步”继续。还可以定制 DEL 选项，如图 6-6 所示。



图 6-6　指定文件位置

(3) 在这一步中，可以指定 LOB 对象的位置。可以指定标识列和生成列行为。另外，可以选择在装载中包含哪些列，如图 6-7 所示。接受默认设置并单击“下一步”。



图 6-7　指定 LOB 对象的位置

(4) 对于索引创建有 3 个选项：递增式、重新构建，或者让 LOAD 自己决定构建索引的最佳方式，如图 6-8 所示。有几个应用程序控制级别，而且 LOAD 有能力使装载后的性能达到最优。



图 6-8　索引、应用程序和性能选项

(5) 在LOAD实用程序运行期间，系统可能会崩溃。为了能够从系统崩溃中恢复，LOAD实用程序提供了崩溃恢复选项，使用户能够指定一致点(savecount)，如图 6-9 所示。因为LOAD 实用程序在事务期间很少进行日志记录，所以有可能需要进行向前恢复。在这一步中，可以选择可恢复(在这里可以保存备份映像的副本)，或者选择不可恢复，从而不允许在发生故障时进行恢复。

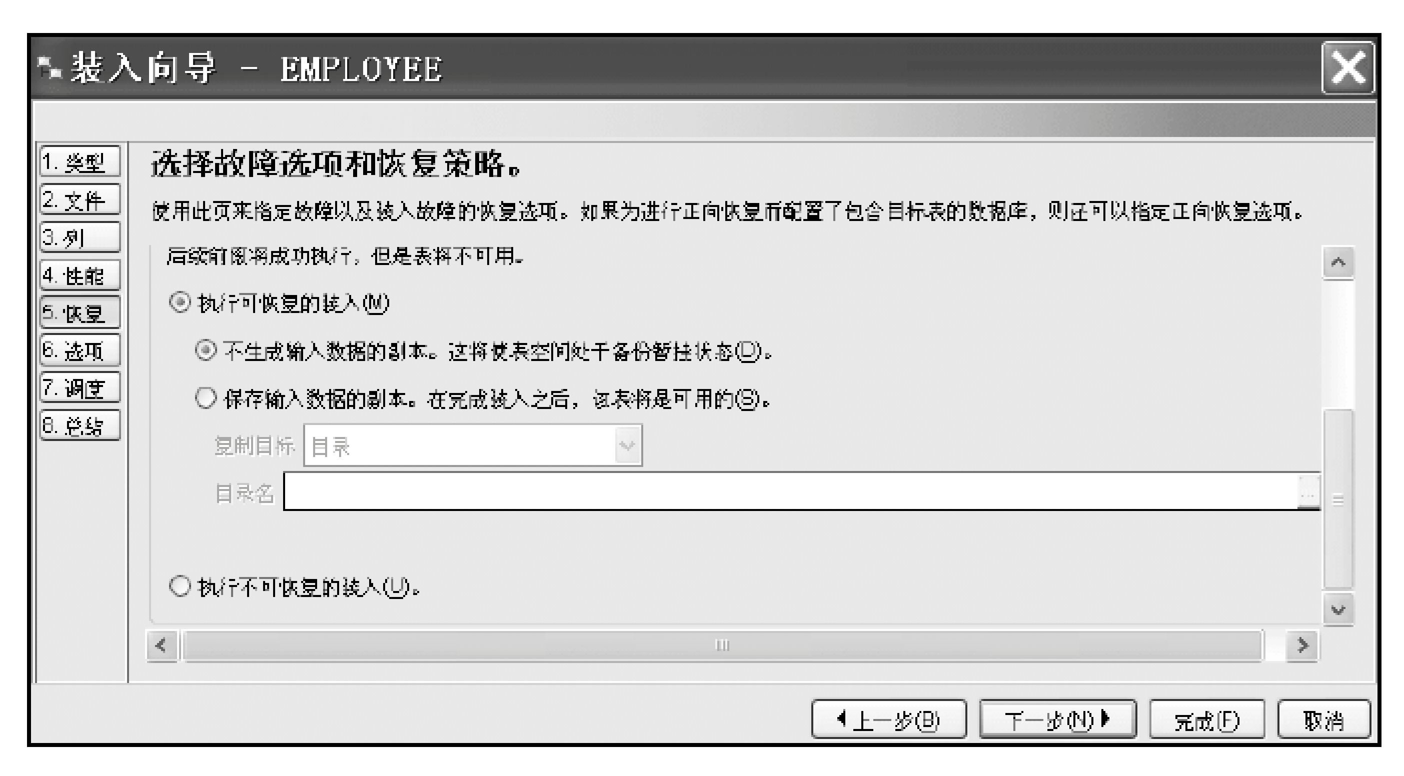


图 6-9　恢复选项

(6) 如果您不能确定合适的性能参数，那么明智的做法是让 DB2 配置顾问进行选择。在这里可以设置的其他选项是异常表的位置，包含被拒绝的行的异常转储文件，以及抑制接收任何警告，如图 6-10 所示。



图 6-10 更多选项

步骤(7)和(8)允许进行调度(就像对 IMPORT 进行调度一样)。最后一步是总结您已经选择的选项。最后(但是并非不重要)，您应该注意 DB2 LOAD 有以下限制：

- 不支持将数据装载进昵称(nickname)、层次结构、有类型表、声明的临时表、包含 XML 列的表或具有结构化类型列的表。
- 如果在 LOAD REPLACE 操作期间发生错误，那么表中原来的数据会丢失。保护措施是保留输入数据的副本，从而允许在发生故障时重新启动装载操作。
- 在新装载的行上不激活触发器。LOAD 实用程序并不实施与触发器相关联的业务规则。

2. 使用 LOAD 命令

```
LOAD [CLIENT] FROM file/pipe/dev/cursor_name [ {,file/pipe/dev} ... ]
     OF {ASC | DEL | IXF | CURSOR}
     [LOBS FROM lob-path [ {,lob-path} ... ] ]
     [MODIFIED BY filetype-mod [ {filetype-mod} ... ] ]]
     [METHOD {L ( col-start col-end [ {,col-start col-end} ... ] )
                 [NULL INDICATORS (col-position [ {,col-position} ... ] )]
            | N ( col-name [ {,col-name} ... ] )
            | P ( col-position  [ {,col-position} ... ] )}]
     [SAVECOUNT n]
     [ROWCOUNT n] [WARNINGCOUNT n] [MESSAGES msg-file]
     [TEMPFILES PATH pathname]
     {INSERT | REPLACE | RESTART | TERMINATE}
     INTO table-name [( insert-column [ {,insert-column} ... ] )]
     [FOR EXCEPTION table-name [NOUNIQUEEXC NORANGEEXC]]
```

```
        [STATISTICS {NO | USE PROFILE}]
        [{COPY {NO  | YES { USE TSM [OPEN num-sess SESSIONS]
                    | TO dir/dev [ {,dir/dev} ... ]
                    | LOAD lib-name [OPEN num-sess SESSIONS]}}
                    | NONRECOVERABLE} ]
        [HOLD QUIESCE] [WITHOUT PROMPTING] [DATA BUFFER buffer-size]
        [SORT BUFFER buffer-size] [CPU_PARALLELISM n] [DISK_PARALLELISM n]
        [INDEXING MODE {AUTOSELECT | REBUILD | INCREMENTAL | DEFERRED}]
        [SET INTEGRITY PENDING CASCADE {DEFERRED | IMMEDIATE}]
        [ALLOW NO ACCESS | ALLOW READ ACCESS [USE tblspace-name]] [LOCK WITH
FORCE]
        [[PARTITIONED DB CONFIG] partitioned-db-option
[{partitioned-db-option}...]]
         filetype-mod:
          NOROWWARNINGS, ANYORDER, BINARYNUMERICS, CODEPAGE=x,
          DUMPFILE=x, FASTPARSE, NOHEADER, TOTALFREESPACE=x,
          INDEXFREESPACE=x, PAGEFREESPACE=x, FORCEIN, IMPLIEDDECIMAL,
          PACKEDDECIMAL, NOCHECKLENGTHS, NOEOFCHAR, NULLINDCHAR=x,
          RECLEN=x, STRIPTBLANKS, STRIPTNULLS, NODOUBLEDEL, LOBSINFILE,
          CHARDELx, COLDELx, DLDELx, DECPLUSBLANK, DECPTx, DATESISO,
          DELPRIORITYCHAR, USEDEFAULTS, DATEFORMAT=x, TIMEFORMAT=x,
          TIMESTAMPFORMAT=x, ZONEDDECIMAL, KEEPBLANKS, IDENTITYMISSING,
          IDENTITYIGNORE, IDENTITYOVERRIDE, GENERATEDMISSING,
          GENERATEDIGNORE, GENERATEDOVERRIDE, USEGRAPHICCODEPAGE
         partitioned-db-option:
          HOSTNAME x, FILE_TRANSFER_CMD x, PART_FILE_LOCATION x,
OUTPUT_DBPARTNUMS x,
          PARTITIONING_DBPARTNUMS x, MODE x, MAX_NUM_PART_AGENTS x,
OMIT_HEADER,
          ISOLATE_PART_ERRS x, STATUS_INTERVAL x, PORT_RANGE x,
CHECK_TRUNCATION,
          MAP_FILE_INPUT x, MAP_FILE_OUTPUT x, TRACE x, NEWLINE, DISTFILE x
```

下面我们来看一个装载示例。

DB2 LOAD 命令文件示例(INSERT)	DB2 LOAD 命令文件示例(REPLACE)
(1)　load	(1)　LOAD
(2)　FROM 'INPUT_FILE1.DAT'	(2)　FROM 'INPUT_FILE2.DAT'
(3)　OF ASC	(3)　OF DEL
(4)　MODIFIED BY DUMPFILE='INPUT_FILE1.BAD'	(4)　MODIFIED BY DUMPFILE='INPUT_FILE2.BAD'
(5)　METHOD L (1 5, 6 15, 16 20)	(5)　COLDEL|

(6) INSERT INTO PROD.TB_TABLE1 (7) (COL1, COL2, COL3) (8) FOR EXCEPTION PROD.TB_ TABLE1_DSC;	(6) CHARDEL" (7) METHOD P (1, 2, 3) (8) REPLACE INTO PROD.TB_TABLE2 (9) (COL1, COL2, COL3) (10) FOR EXCEPTION PROD.TB_TABLE2_DSC;

下面我们解释上面例子中用到的命令选项：

(1) LOAD

这会在 DB2 中调用 LOAD 实用程序。

(2) FROM [inputfile_name]

这是包含要装载的数据的文件。DB2 LOAD 还可以从管道、设备或游标装载数据。

(3) OF ASC/ DEL

对于 DB2 LOAD，ASC 表示不分界的 ASCII 数据，数据的划分由位置决定。DEL 表示分界的 ASCII 数据，每行的数据长度可变。分界的数据可以使用多种修饰符，主要的两种是 COLDEL 和 CHARDEL；COLDEL 决定列和列之间如何分界，默认设置是逗号；CHARDEL 决定字符串数据如何分界，默认设置是双引号。

(4) MODIFIED BY DUMPFILE=[dumpfile_name]

DB2 把被拒绝的记录放到这个文件中。

(5) METHOD P (1，2，3)

DB2 LOAD 有 3 个方法：

- METHOD L 只用于 ASC 数据，这个方法要指出每列的开头和结尾。形式是：METHOD L(start1 end1，start2 end2….)。
- METHOD N 用于 IXF 或游标数据，这个方法要指定源表中要装载的列。形式是：METHOD N(col1，col2，col4…)。
- METHOD P 用于 DEL、IXF 或游标数据，这个方法要指定源数据中要装载的列的位置(position)号。形式是：METHOD P (1，2，4…)。

(6) INSERT / REPLACE INTO PROD.TABLE

DB2 LOAD 在这里有两个选项：INSERT 和 REPLACE。另外两个 DB2 LOAD 选项是 RESTART 和 TERMINATE。当 DB2 LOAD 由于任何原因未完成时，使用这些选项。

(7) (COL1，COL2，COL3) Insert Column List

DB2 LOAD 使用这个列表决定要放入数据的列。如果省略这个列表，那么 DB2 LOAD 会尝试按照读取和解析数据的次序装载数据。

(8) FOR EXCEPTION [table_name]

DB2 LOAD 把违反唯一约束和主键规则的记录(异常)放到以前创建的这个表中。

LOAD 命令非常复杂，基本上是 DB2 中最复杂的命令，有很多命令选项，表 6-4 总结了常用的命令选项。还有一些选项我们会在后面特定的 LOAD 性能中讲到。

表 6-4　常用的 LOAD 命令选项

LOAD 命令关键字	LOAD 命令关键字解释
WARNINGCOUNT=number	使用此参数来指定强制装入操作终止之前 LOAD 实用程序可返回的警告数目。如果只要很少警告或不需要警告，那么将 WARNINGCOUNT 参数设置为相对较小的数字。装入操作将在达到 WARNINGCOUNT 数目时停止。这允许您在尝试完成装入操作之前解决问题
NOROWWARNINGS	在装入操作期间，关于已拒绝的行的警告消息将写入指定的文件中。但是，如果 LOAD 实用程序必须处理大量已拒绝的、无效或已截断的记录，那么可能会对装入性能产生负面影响。如果预计到会产生许多警告，那么使用 NOROWWARNINGS 文件类型修饰符来抑制记录这些警告会很有用。但是这样做可能有少许风险
MESSAGES messagefile	DB2 把消息放到这个消息文件中。如果不指定消息文件，就不产生消息。为了定位导入期间产生的问题，建议指定消息文件
ROWCOUNT number	指定要装载的记录数。如果省略这个关键字，那么默认设置是所有记录
CPU_PARALLELISM number DISK_PARALLELISM number FETCH_PARALLELISM yes	DB2 V9 中 LOAD 自动决定这些设置，用来控制对文件、设备、管道和游标装载中的记录进行解析、转换、格式化和写操作所生成的线程数。也可以使用这些关键字指定自己需要的值。可以提高性能
DATA BUFFER number	LOAD 使用许多大小为 4KB 的页面传输数据，这个数值通常是自动决定的，但是也可以用这个关键字指定自己需要的值
SAVECOUNT number	LOAD 使用一致点确保装载操作的可恢复性
RESTART REPLACE， INSERT， TERMINATE	LOAD 使用 RESTART 模式在遇到故障之前的最后一个一致点之后选择重新装载的起始点。LOAD 会自己决定起始点，不需要操作者计算。REPLACE 覆盖已有数据、INSER 追加数据、TERMINATE 终止 LOAD 程序

(续表)

LOAD 命令关键字	LOAD 命令关键字解释
INDEXING MODE DEFERRED	该模式意味着在装载期间不会创建索引。涉及的索引上会作出标记，但是需要刷新。当重新启动数据库或者第一次访问那些索引时，才会重新构建那些索引
INDEXING MODE REBUILD，INCREMENTAL，AUTOSELECT	REBUILD 模式强制重新构建所有索引，INCREMENTAL 模式只向索引中添加新增的数据，AUTOSELECT 模式允许实用程序在 REBUILD 和 INCREMENTAL 之间作出选择
NONRECOVERABLE 和 COPY NO 区别	如果启用前滚恢复，那么在前滚后不需要对表恢复装入事务的情况下使用此参数。NONRECOVERABLE 装入和 COPY NO 装入具有完全相同的性能。但是，在潜在数据丢失方面却有重大差别。NONRECOVERABLE 装入将表标记为不可前滚恢复，并同时使得能够完全访问表。这可能会产生一个问题，在需要前滚装入操作的情况下，已装入的数据以及对表的所有后续更新都会丢失。COPY NO 装入使所有从属表空间处于“备份暂挂”状态，这将导致在执行备份之前，表不可访问。因为在该类型的装入后会强制执行表空间备份，所以不存在丢失已装入数据或对表的后续更新的风险。也就是说，COPY NO 装入完全可恢复。如果使用这个选项，在装载操作之后表空间处于 backup pending 状态，在装载操作期间不复制(copy)装载的数据
COPY YES	使用此参数来指定在装入操作期间是否创建输入数据的副本。仅当启用了归档日志时，COPY YES 才适用。COPY NO 选项会导致与装入的表相关的表空间将处于“备份暂挂”状态，并且必须先备份这些表空间才能访问该表。而 COPY 在 LOAD 期间自动做表空间备份
MODIFIED BY RECLEN=x	固定的输入记录的大小。适用于 ASC 文件格式
MODIFIED BY DUMPFILE=filename	指定一个文件，用来决定在哪里存储被拒绝(reject)的记录
FOR EXCEPTION tablename	异常表存储不遵循唯一约束和主键约束的行。如果装入的时候没有指定异常表，违反唯一约束的行将被丢弃并且不再有机会恢复或修改。异常表结构比要装入表的结构增加了两列：时间戳列和消息描述列 CLOB(32KB)
OF DEL，ASC，IXF 和 WSF	LOAD 数据格式

(续表)

LOAD 命令关键字	LOAD 命令关键字解释
INTO TABLE tablename	要存储数据的目标表
MODIFIED BY COLDELx	在所有输入数据中以这个字符分隔各个列(默认设置是逗号)
MODIFIED BY CHARDELx	用这个字符包围输入的字符数据(默认设置是双引号)
1. LOBS FROM pathnames 2. MODIFIED BY LOBSINFILE	寻找 LOB 文件的路径。要想使用 LOBS FROM 子句，就必须设置这个关键字。参数本身包含文件名，但是不包含完整的路径名，因为将搜索 LOBS FROM 路径
ALLOW NO ACCESS/ ALLOW READ ACCESS	在装入操作执行过程中，是否可以查询表中预先存在的数据。ACCESS READ ACCESS 选项允许在进行装入操作时查询表。只能查看装入操作之前表中已存在的数据
LOCK WITH FORCE	装入操作在继续执行之前是应该等待其他实用程序或应用程序使用完表，还是应该强制其他应用程序释放锁定
STATISTICS USE PROFILE	在装入过程中是否收集统计信息。仅当以 REPLACE 方式运行装入操作时，才支持此选项
TEMPFILES PATH	装入操作执行期间创建临时文件时要使用的标准路径。此名称由 LOAD 命令的 TEMPFILES PATH 参数指定。默认值是数据库路径，并且由 DB2 实例以独占方式访问
USE <tablespace-name>	如果正在执行 ALLOW READ ACCESS 装入并且建立索引方式为 REBUILD，那么此参数允许在系统临时表空间中重建索引，并在装入操作的索引复制阶段将其复制回索引表空间

3. 使用 ADMIN_CMD 存储过程

可以在命令行中直接调用 ADMIN_CMD 存储过程以导出数据，如下所示：

```
    call sysproc.admin_cmd('load from /home/db2inst1/output/sales.del of del
messages /home/db2inst1/output/load.msg replace into sales')
```

6.4.3　装入示例

例 6-9　看看下面这个例子，它演示了装载过程中涉及的步骤：

```
    LOAD FROM emp.ixf OF IXF    MESSAGES msg.out    MODIFIED BY
DUMPFILE=c:\emp.dmp
```

```
TEMPFILES PATH d:\tmp     INSERT INTO employee     FOR EXCEPTION empexp
```

在图 6-11 中：

(1) 显示了输入源文件的内容。

(2) 目标表 EMPLOYEE 是用以下列定义创建的：第一列必须是唯一的。最后一列是数值列且不能为 NULL。

(3) 异常表 EMPEXP 是使用和 EMPLOYEE 相同的列，再加上时间戳和消息列创建的。

在装载阶段，输入文件中的所有数据被装载到 EMPLOYEE 中，除了标为粉色的两个行(20 和 40)，因为它们不符合 NOT NULL 和 NUMERIC 列定义。由于指定了 DUMPFILE 修饰符，因此这两行的数据被记录在 C:\emp.dmp 文件中。

在删除阶段，标为黄色的两个行(30 和 50)被从 EMPLOYEE 中删除，并插入到异常表 EMPEXP 中。这是因为它们违反了 EMPLOYEE 表中第一列的唯一性约束。

在装载的最后，应该检查消息文件、转储文件和异常表，然后决定如何处理被拒绝的行。如果装载成功完成，那么在 D:\tmp 中生成的临时表将自动被删除。

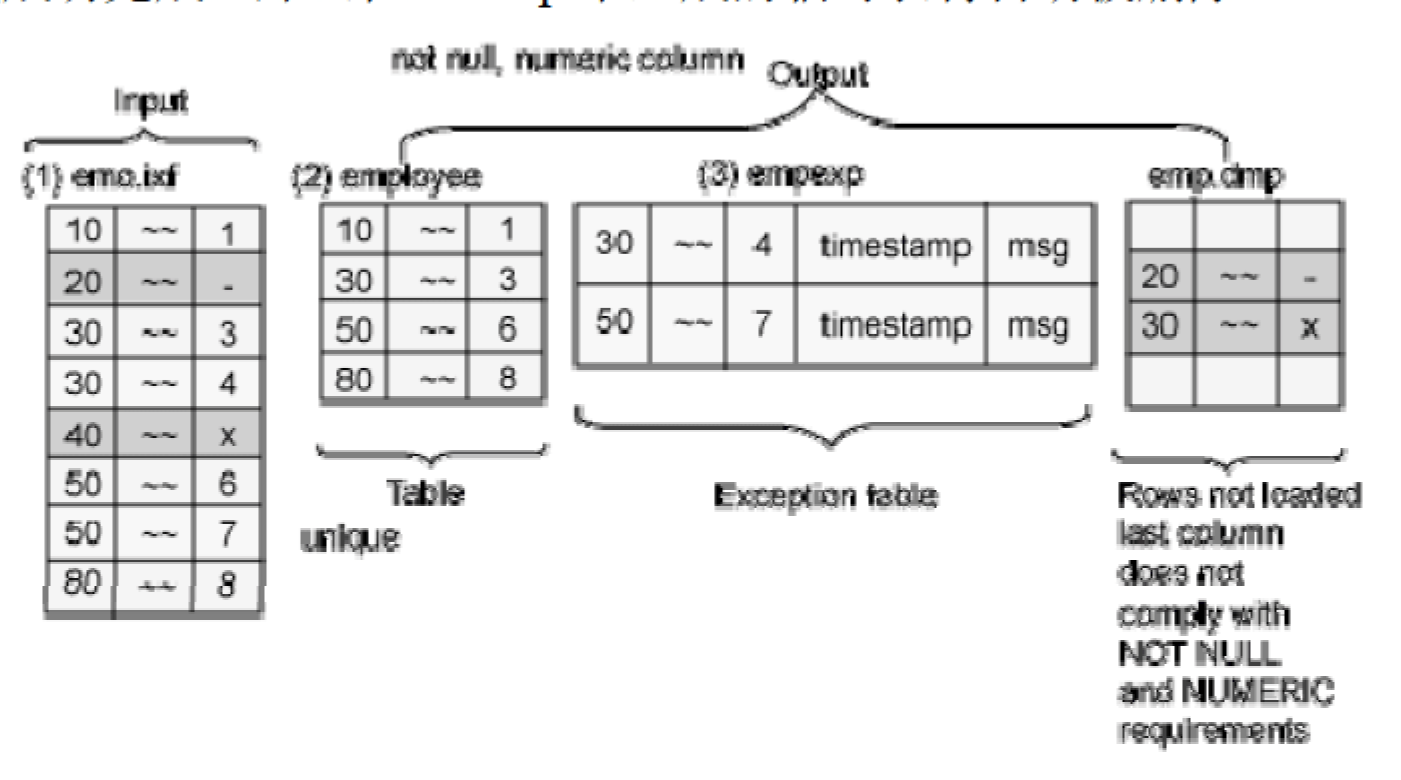


图 6-11　装载过程

例 6-10　使用转储文件。

表 FRIENDS 的定义如下所示：

```
create table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

如果尝试将下列数据记录装入到表中：

```
23, 24, bobby
, 45, john
4,, mary
```

将拒绝第二行，这是因为第一个 INT 是 NULL，但列定义指定了 NOT NULL。包含与 DEL 格式不一致的初始字符的列将生成错误，将拒绝该记录。您可以将此类记录写入转储文件。

在第一列中，将忽略字符定界符外部的 DEL 数据，但会生成警告。例如：

```
22,34,"bob"
24,55,"sam" sdf
```

LOAD 实用程序将在表的第三列中装入"sam"，并且将会在一条警告消息中标记字符"sdf"。不会拒绝该记录。另一个示例：

```
22 3, 34,"bob"
```

LOAD 实用程序将装入 22、34、"bob"并生成一条警告消息，该消息指出忽略了第一列中 22 后面的某些数据。不会拒绝该记录。

例 6-11　从游标装入。

假定源表和目标表位于同一数据库中，并且它们的定义如表 6-5 所示。

表 6-5　源表和目标表的结构

表 DEV.TABLE1 有 3 列：结构如下	表 DEV.TABLE2 有 3 列：结构如下
ONE INT	ONE INT
TWO CHAR(10)	TWO CHAR(10)
THREE DATE	THREE DATE

游标 MYCURSOR 是按以下方式定义的：

```
declare mycursor cursor for select * from dev.table1
```

以下命令将 MY. TABLE1 中的所有数据装入到 MY. TABLE2 中：

```
load from mycursor of cursor method P(1,2,3)  insert into  dev.table2(one,two,three)
```

注意：

① 在单个 LOAD 命令中只指定了一个游标名。也即，不允许 load from mycurs1、mycurs2 of cursor...。

② 对于从游标装入来说，有效的 METHOD 值只有 P 和 N。

③ 在此例中，由于插入列列表(one、two、three)表示默认值，因此可以将 METHOD P 和该插入列列表省略。

④ MY. TABLE1 可以是表、视图、别名或昵称。

例 6-12 限制装入行数。

用 ROWCOUNT 选项可以指定从文件开始处装入的记录数。例如，导入前 3 条记录：

```
LOAD FROM D:\STAFF.TXT OF DEL ROWCOUNT 3INSERT INTO STAFF1
```

注意：

staff. txt 文件是我们从表 staff 中导出的 DEL 格式的文件。

例 6-13 出现警告信息时强令装入操作失败。

在某些情况下，文件中的数据必须全部成功输入到目标表中才算成功，即使有一条记录出错也不行。在这种情况下，可以使用 WARNINGCOUNT 选项。把输入文件 staff. txt 中第一列数据类型为数字并等于 320 的行替换为："abcf"，"aaa"，"sdfg"。执行下列命令：

```
LOAD FROM D:\STAFF.TXT OF DEL WARNINGCOUNT 1 INSERT INTO STAFF1
```

运行结果包含下面的警告：

```
SQL3118W  在行 "32" 列 "1"中的字段值不能转换为 SMALLINT 值，但是目标列不可为空。未
          装入该行。
SQL3185W  当处理输入文件的第 "32" 行中的数据时发生先前的错误。
SQL3502N  实用程序遇到了"1" 个警告，它超过了允许的最大警告数。
```

此时无法对表 STAFF1 进行操作，例如：

```
SELECT * FROM STAFF1
```

会返回：

```
ID     NAME      DEPT   JOB   YEARS  SALARY    COMM
------ --------- ------ ----- ------ --------- ---------
SQL0668N  由于表 "USER.STAFF1" 上的原因代码 "3"，因此不允许操作。
SQLSTATE=57016
```

原因是：表处于“装入挂起”状态。对此表的先前的 LOAD 尝试失败。在重新启动或终止 LOAD 操作之前不允许对表进行存取。

解决方法为：通过分别发出带有 RESTART 或 TERMINATER 选项的 LOAD 来重新启动或终止先前失败的对此表的 LOAD 操作。

包含 TERMINATER 的 LOAD 命令可以终止装入进程，使目标表恢复正常可用状态：

```
LOAD FROM D:\STAFF.TXT OF DEL TERMINATE INTO STAFF1
```

包含 RESTART 的 LOAD 命令可以在源文件修改正确的时候使用，使装入进程重新开始：

```
LOAD FROM D:\STAFF.TXT OF DEL RESTART INTO STAFF1
```

例 6-14　防止产生警告信息。

使用 NOROWWARNINGS 文件类型修饰符可以禁止产生警告信息，当装入过程可能出现大量警告信息，而用户对此又不感兴趣时，可以使用该选项，这样可以大大提高装入的效率。打开 staff. txt 文件，把第一列等于 320 的行替换为："abcf"，"aaa"，"sdfg"。

```
LOAD FROM D:\STAFF.TXT OF DEL MODIFIED BY NOROWWARNINGS INSERT INTO STAFF
```

运行结果中，第 32 行出错，该行无法装入，但是不产生警告信息。

例 6-15　生成统计数据。

使用 STATISTICS 选项可以在装入的过程中生成统计数据，这些统计数据可以供优化器确定最有效的执行 SQL 语句的方式。可以对表和索引产生不同详细程度的统计数据。

对表和索引产生最详细的统计数据：

```
LOAD FROM D:\STAFF.TXT OF DEL REPLACE INTO STAFF1 STATISTICS YES WITH D
ISTRIBUTION  AND DETAILED INDEXES ALL
```

对表和索引都产生简略的统计数据：

```
LOAD FROM D:\STAFF.TXT OF DEL REPLACE INTO STAFF1 STATISTICS YES AND IN
DEXES ALL
```

6.4.4　在线 LOAD

在 DB2 V8 之前，当表被装载时，LOAD 实用程序使用超级排它锁将表锁定。并且 LOAD 所在的表空间也处于 LOAD PENDING 状态而无法访问。在 DB2 V8 以后，为了增强 LOAD 的可用性，我们可以指定 LOAD 的选项 ALLOW READ ACCESS。如果没有指定该选项，默认是 ALLOW NO ACCESS，此选项在装载完成之前，不能访问正在装载的数据。这个选项使正在装载数据的表处于 LOAD IN PROGRESS 状态和 READ ACCESS ONLY 状态。ALLOW READ ACCESS 选项导致被装载的表以共享的方式锁定。我们可以访问表中已有的数据，但是不能访问新装载的那部分数据。图 6-12 是在线和离线 LOAD 的示意图。

通过图 6-12 我们可以看到 ALLOW NO ACCESS 选项以独占方式锁定表，在装入该表时不允许对表数据进行访问，这个锁直到 LOAD 完成才释放。

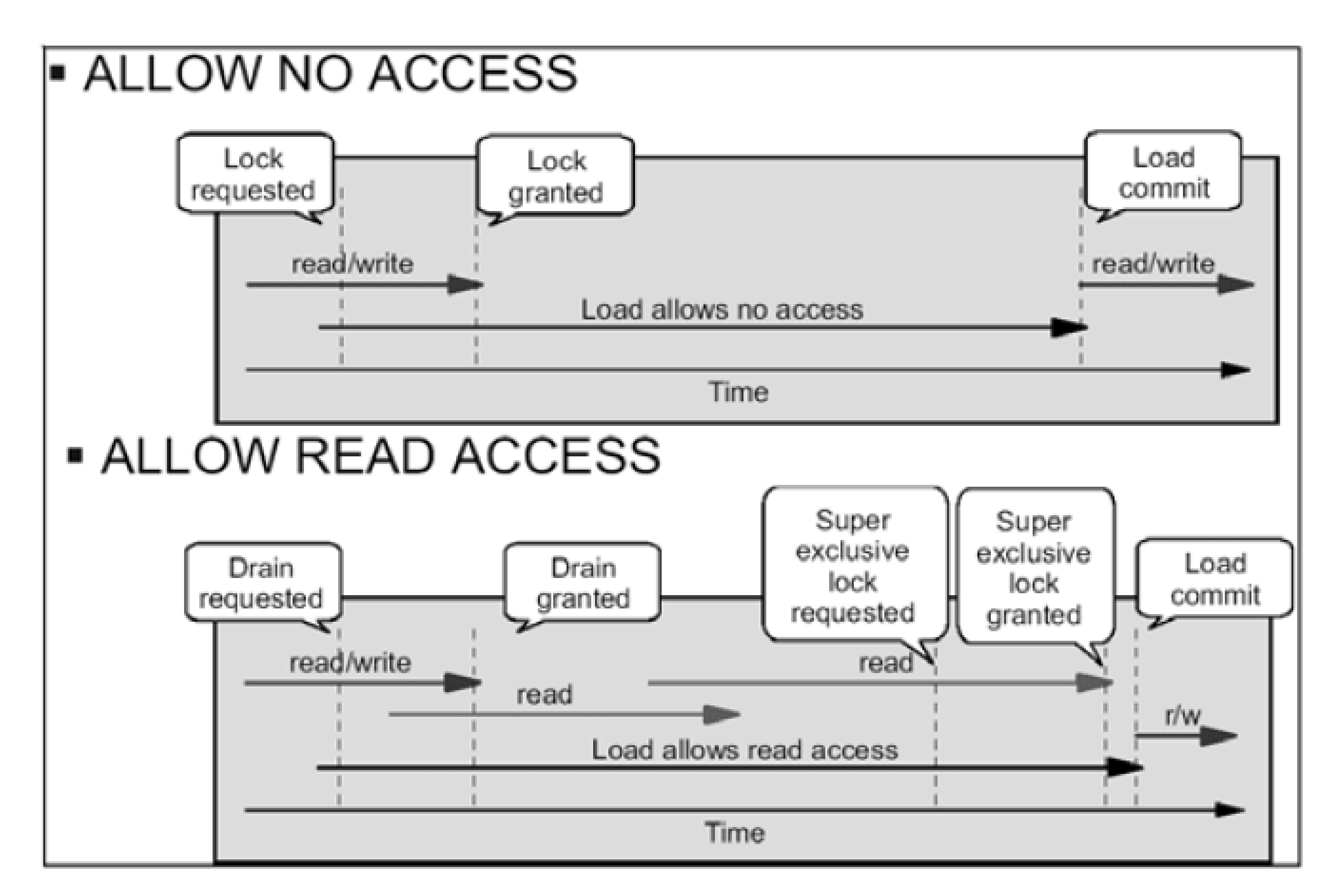


图 6-12　在线和离线 LOAD

对于 ALLOW READ ACCESS 选项我们看到，当我们开始 LOAD 时，如果某个应用程序已经锁住了目标表，那么 LOAD 实用程序就必须等到这些锁被释放。为了不必等到一个锁释放出来，可以使用 LOAD 命令中的 LOCK WITH FORCE 选项，使持有冲突锁的其他应用程序强制离开。ALLOW READ ACCESS 选项不允许其他应用程序对该表进行任何写访问，但允许对预先存在的数据进行读访问。在装入操作的整个执行过程中，除了操作开始和操作结束时以外，都允许进行读访问。下面我们详细讲解 ALLOW READ ACCESS 访问机制。

首先，在设置阶段接近结束时，装入操作将获取特殊的 Z 锁定并占用一小段时间。如果某个应用程序在装入操作请求这个特殊的 Z 锁定前，对该表挂起了不兼容的锁定，那么装入操作在发生超时和失败之前将等待一段有限的时间，以允许这个不兼容的锁定被释放。等待时间长度由 *locktimeout* 数据库配置参数确定。如果指定 LOCK WITH FORCE 选项，装入操作就会强制其他应用程序释放锁定以避免超时。装入操作获取特殊的 Z 锁定、落实设置阶段、释放该锁定并进入装入阶段。在 ALLOW READ ACCESS 方式的装入操作启动后，任何对该表请求读访问锁定的应用程序都将获得该锁定，而不会与这个特殊的 Z 锁定发生冲突。尝试读取目标表中现有数据的新应用程序也能够成功地完成操作。

其次，在装入操作结束时，LOAD 实用程序在落实数据之前对该表获取互斥锁定(Z 锁定)。LOAD 实用程序将等待所有对该表挂起了锁定的应用程序释放那些锁定。这会导致落实数据前发生延迟。LOCK WITH FORCE 选项用来强制有冲突的应用程序释放锁定，这样装入操作就能够继续执行而不必等待。通常，ALLOW READ ACCESS 方式的装入操作获取互斥锁定并占用一小段时间；但是，如果指定 USE <tablespace-name>选项，就会在整个

索引复制阶段占用互斥锁定。

在装入操作执行过程中，可以对在装入操作启动前存在的表数据和索引数据进行查询。下面我们用一个例子来说明。

```
 创建包含一个整数列的表：
     create table t1 (id int)
 装入三行：
     load from t1.txt of del insert into t1   ...
     读取的行数                       = 3
     跳过的行数                       = 0
     已装入的行数                     = 3
     拒绝的行数                       = 0
     删除的行数                       = 0
     已落实的行数                     = 3
查询该表：
     select * from t1
                    ID
                 -----------
                     1
                     2
                     3
    3 record(s) selected.
在指定了 ALLOW READ ACCESS 选项的情况下执行装入操作并装入另外两行数据：
     load from t12.txt of del insert into t1 allow read access
同时，使用另一个连接，在装入操作执行过程中查询该表：
     select * from t1
                    ID
                  -----------
                     1
                     2
                     3
    3 record(s) selected.
等待装入操作完成，然后查询该表：
     select * from t1
                    ID
                 -----------
                     1
                     2
                     3
                     4
                     5
    5 record(s) selected.
```

由于 ALLOW READ ACCESS 选项允许用户在任何时间(甚至在装入操作执行时或者在装入操作失败后)访问表数据，因此在装入大量数据时，此选项非常有用。在 ALLOW READ ACCESS 方式下，装入操作的行为独立于应用程序的隔离级别。换言之，具有任何隔离级别的应用始终能够读取预先存在的数据，但它们在装入操作完成前无法读取新装入的数据。

对于 ALLOW READ ACCESS 选项，如果重新构建完整的索引，那么将创建索引的影子副本。当 LOAD 实用程序进入索引复制阶段(见装载过程的 4 个阶段)时，目标表将离线，新的索引被复制到目标表空间。

无论指定哪种表访问选项，装载操作都需要得到各种不同的锁才能继续。

6.4.5 监控 LOAD 进度

LOAD QUERY

当表被装载时，LOAD 实用程序使用排它锁将表锁定。在装载完成之前，对表的其他访问是不允许的。这是 ALLOW NO ACCESS 选项的默认行为。在那样的装载期间，表处于 LOAD IN PROGRESS 状态。如果我们想查看 LOAD 的进度，可以通过 LOAD QUERY 命令检查装载操作的状态和返回表状态：

```
LOAD QUERY TABLE shcemaname.table_name
```

下面我们举一个使用 LOAD QUERY 查询 LOAD 进度的例子。

```
C:\>db2 load query table oracle.dept1
SQL3109N  实用程序正在开始从文件 "C:\dept.txt" 装入数据。
SQL3500W  在时间 "2008-11-23 00:56:21.772682"，实用程序在开始 "LOAD"。
SQL3519W  开始装入一致点。输入记录数 = "0"。
SQL3520W  “装入一致点”成功。
SQL3110N  实用程序已完成处理。从输入文件读了"229376" 行。
SQL3519W  开始装入一致点。输入记录数 = "229376"。
SQL3520W  “装入一致点”成功。
SQL3515W  在时间 "2008-11-23 00:56:23.415390"，实用程序已经完成了"LOAD"。
SQL3532I  Load 实用程序当前正处于"LOAD" 阶段。
读取行数          = 229376
跳过行数          = 0
装入行数          = 229376
拒绝行数          = 0
删除行数          = 0
落实行数          = 229376
警告数            = 0
表状态:
  正在装入(load in progress)
C:\>
```

LIST UTILITIES

调用 LOAD 实用程序后，除了使用 LOAD QUERY 查询 LOAD 进度外，我们还可以使用 LIST UTILITIES 命令来监视装入操作的进度。在以 INSERT 方式、REPLACE 方式或 RESTART 方式执行装入操作时，将提供详细进度监视支持。发出带有 SHOW DETAILS 选项的 LIST UTILITIES 命令来查看关于当前装入阶段的详细信息。以 TERMINATE 方式执行装入操作时，将无法获取详细信息。LIST UTILITIES 命令将仅仅显示装入终止实用程序当前正在运行，下面我们举一个使用 LIST UTILITIES 监控 LOAD 进度的例子。

```
C:\>db2 list utilities show detail
标识                                   = 7
类型                                   = LOAD
数据库名称                             = SAMPLE
分区号                                 = 0
描述        = OFFLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO ORACLE  .DEPT1
开始时间                 = 2008-11-23 01:01:50.423478
状态                     = 执行
调用类型                 = 用户
进度监视:
   阶段号                          = 1
      描述                         = SETUP
      总计工作                     = 0 bytes
      已完成的工作                 = 0 bytes
      开始时间                     = 2008-11-23 01:01:50.423489
   阶段号 [当前]                    = 2
      描述                                         = LOAD(构建阶段)
      总计工作                     = 226508 rows
      已完成的工作                 = 170081 rows
      开始时间                     = 2008-11-23 01:01:50.524716
```

6.4.6　LOAD 期间和之后的表空间状态

在装入操作期间，LOAD 实用程序使用表空间状态来保持数据库的一致性。这些状态通过控制对数据的访问或引发用户操作来起作用。

可以使用 LIST TABLESPACES、LOAD QUERY 和 LIST UTILITIES 命令来检查表空间状态。表空间可以同时处于多种状态。LOAD 期间和之后的常见状态有：

正常(normal)

“正常”状态是创建表空间后该表空间的初始状态，它指示当前没有(异常)状态影响表空间。

只读访问(read access)

如果指定 ALLOW READ ACCESS 选项，那么表将处于“只读访问”状态。在调用 LOAD 命令前存在的表数据在装入操作运行期间可供只读访问。如果指定 ALLOW READ ACCESS 选项并且装入操作失败，那么在装入操作前存在的表数据在故障发生后将继续可供只读访问。

正在装入(load in progress)

“正在装入”状态指示正在表空间上进行装入。此状态不允许在装入操作期间备份从属表。“正在装入”表空间状态与“正在装入”表状态(所有装入操作中都使用此状态)不同，因为仅当对可恢复数据库指定了 COPY NO 参数时，LOAD 实用程序才使表空间处于“正在装入”状态。

将包含大量数据的输入文件(staffdata.del)装入到 NEWSTAFF 表中，如下所示：

```
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff allow read access;
```

打开另一个会话并发出下列命令：

```
connect to sample;
load query table newstaff;
```

LOAD QUERY 命令将显示 NEWSTAFF 表处于“只读访问”和“正在装入”状态：

```
表状态:
  正在装入(load in progress)
  只读访问(READ ACCESS)
```

备份暂挂(backup pending)

如果对可恢复数据库执行装入操作并且指定 COPY NO 参数，那么在第一次落实后表空间将处于“备份暂挂”表空间状态。不能更新处于“备份暂挂”状态的表空间。通过备份表空间即可使表空间脱离“备份暂挂”状态。由于装入操作开始时会更改表空间状态并且不能回滚，因此即使取消装入操作，表空间也保持处于“备份暂挂”状态。

将输入文件(staffdata.del)装入到 NEWSTAFF 表中，如下所示：

```
update db cfg for sample using logretain recovery;---更改为归档日志
backup db sample;
connect to sample;
```

```
create table newstaff like staff;
load from staffdata.del of del insert into newstaff copy no;
```

打开另一个会话并发出下列命令：

```
connect to sample;
list tablespaces;
```

那么 USERSPACE1(样本数据库的默认表空间)将处于“正在装入”状态，并且在第一次提交后，将处于“备份暂挂”状态。在装入操作完成后，LIST TABLESPACES 命令表明 USERSPACE1 现在处于“备份暂挂”状态：

```
表空间标识                     = 3
名称                           = IBMDB2SAMPLEREL
类型                           = 数据库管理空间
内容                           = 所有持久数据。大型表空间。
状态                           = 0x0020
  详细解释：
    备份暂挂(backup pending)
```

复原暂挂(restore pending)

如果使用 COPY NO 选项成功执行了装入操作、复原数据库，然后前滚该操作，那么相关表空间将处于“复原暂挂”状态。要使表空间脱离“复原暂挂”状态，必须执行复原操作。

装入暂挂(load pending)

“装入暂挂”表状态指示装入操作失败或被中断。可以执行以下其中一种方法来除去“装入暂挂”状态：

- 找出故障原因。例如，如果 LOAD 实用程序耗尽了磁盘空间，那么对表空间添加容器。然后，重新启动(restart)装入操作。
- 终止(terminate)装入操作。
- 对装入操作失败时处理的那个表进行 LOAD REPLACE 操作。
- 使用最新的表空间或数据库备份，通过 RESTORE DATABASE 命令恢复装入的表的表空间，然后执行进一步的恢复操作。

不可重新启动装入

处于“不可重新启动装入”状态时，表已部分装入，并且不允许装入重新启动操作。在下面两种情况下，表会处于“不可重新启动装入”状态：

- 在未能成功地重新启动或终止的失败装入操作后，执行前滚操作。
- 根据表处于“正在装入”或“装入暂挂”状态时创建的联机备份执行复原操作。

表还将处于“装入暂挂”状态。要使表脱离“不可重新启动装入”状态，发出 LOAD TERMINATE 或 LOAD REPLACE 命令。

设置完整性暂挂

“设置完整性暂挂”状态指示已装入的表有未经验证的约束。当 LOAD 实用程序开始对带有约束的表执行装入操作时，就会使表处于此状态。使用 SET INTEGRITY 语句以使表脱离“设置完整性暂挂”状态。下面我们举一个设置完整性暂挂的例子：

1. Connect to sample　　　　　　　　　　连接到 sample 数据库

2. CREATE TABLE STAFF1 LIKE STAFF　　　　创建一个结构与 STAFF 表相同的表：

3. alter table staff1 add constraint chk check(dept<100) 给该表添加检查约束

4. LOAD FROM D:\STAFF.TXT OF DEL INSERT INTO STAFF1，打开 STAFF.TXT 文件，把最后一行数据的第三列改为 150，这样这条数据就不满足第 3 步加上的检查约束条件了，然后用 LOAD 命令从文件中装入数据到 staff1 表中

5. Select * from staff1

此时运行查询命令，会得到错误信息：SQL0668N 由于表 "USER.STAFF1" 上的原因代码 "1"，因此不允许操作。SQLSTATE=57016 原因是装入时有数据违反了检查约束，造成表处于检查挂起状态。

6. set integrity for staff1 check immediate unchecked 解除表的检查挂起状态，使用：再次运行查询命令

Select * from staff1

发现表可以正常使用了，其中的违反检查规则的数据也存在。

不可用

通过不可恢复的装入操作执行前滚将使表处于“不可用”状态。处于此状态时，表不可用；必须 drop 该表或通过备份复原表。

6.4.7 使用 CURSOR 文件类型移动数据

通过在使用 LOAD 命令时指定 CURSOR 文件类型，可以将 SQL 查询结果直接装入到目标表中，而不必创建中间导出的文件。

有 3 种方法可用于通过使用 CURSOR 文件类型来移动数据。第 1 种方法是使用命令行处理器(CLP)，第 2 种方法是使用 API，第 3 种方法是使用 ADMIN_CMD 过程。要从 CLP

中执行 LOAD FROM CURSOR 操作，首先必须对 SQL 查询声明游标。一旦声明游标，就可以发出 LOAD 命令，并将所声明游标的名称用作 *cursorname*，将 CURSOR 用作文件类型。

例 6-16 假定源表和目标表位于同一数据库中，并且它们的定义如表 6-6 所示。表上没有索引，现在我们要把表 T1 的数据导入到表 T2 中，数据量大概为 6 千万条记录。图 6-13 对 CUSOR 方式和其他方式进行了比较。

表 6-6 源表和目标表的结构

表 DEV.T1 有 3 列：结构如下	表 DEV.T2 有 3 列：结构如下
ONE INT	ONE INT
TWO CHAR(10)	TWO CHAR(10)
THREE DATE	THREE DATE

Operation(s)	Elapsed Time
INSERT INTO T2 SELECT * FROM T1 (with logging)	1838 seconds
INSERT INTO T2 SELECT * FROM T1 (using not logged initially)	1818 seconds
EXPORT TO <file> OF IXF SELECT * FROM T1 LOAD FROM <file> OF IXF INSERT INTO T2	99 seconds (31s + 68s)
EXPORT TO <pipe> OF IXF SELECT * FROM T1 LOAD FROM <pipe> OF IXF INSERT INTO T2 (concurrently)	68 seconds
DECLARE CURSOR1 CURSOR FOR SELECT * FROM T1 LOAD FROM CURSOR1 OF CURSOR INSERT INTO T2	68 seconds
EXPORT TO <file> OF IXF SELECT * FROM T1 IMPORT FROM <file> OF IXF INSERT INTO T2	1940 seconds (32s + 1908s)

图 6-13 CUSOR 和其他方式的比较

执行下列 CLP 命令会将 DEV. T1 中的所有数据装入到 DEV. T2 中：

```
c:>db2 +c  DECLARE mycurs CURSOR FOR SELECT TWO, ONE, THREE FROM dev.t1
c:>db2 +c  LOAD FROM mycurs OF cursor INSERT INTO dev.t2  ---+c 是关闭自动提交
```

从图 6-13 中我们可以看出通过游标方式装载数据花费时间最小。过去没有 CUOSR 方式时，我们需要把数据落地(图 6-13 中显示落地花了 31s)，然后再装载。而现在可以把这个落地步骤省掉，从而提高 LOAD 速度。

6.4.8 提高 LOAD 性能

在众多数据库中，LOAD 装载速度最快。但是我们还是可以使用各种命令参数来优化装入性能。还有许多对装入来说唯一的文件类型修饰符，在某些情况下，这些修饰符可以极大地提高 LOAD 实用程序的性能。

1. 并行性和装入

LOAD 实用程序使用多个处理器或多个存储设备的硬件配置，如对称多处理器(SMP)环境。

通过使用 LOAD 实用程序，可以有多种方法来并行处理大量数据。一种方法是使用多个存储设备，这允许在装入操作期间利用 I/O 并行性(图 6-14 所示)。另一种方法涉及在 SMP 环境中使用多个处理器，这允许利用分区内并行性(图 6-15 所示)。这两种方法可一起使用以提高数据装入速度。

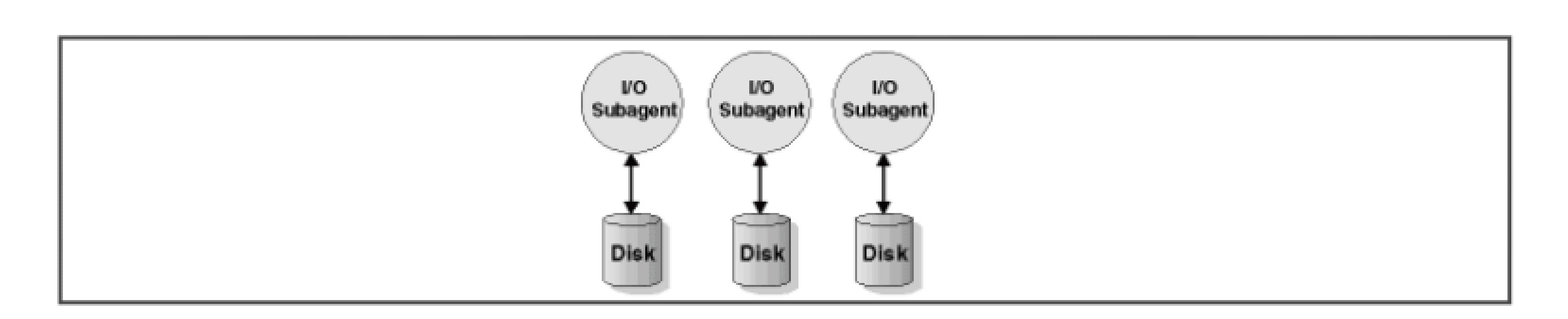


图 6-14　在装入数据时利用 I/O 并行性

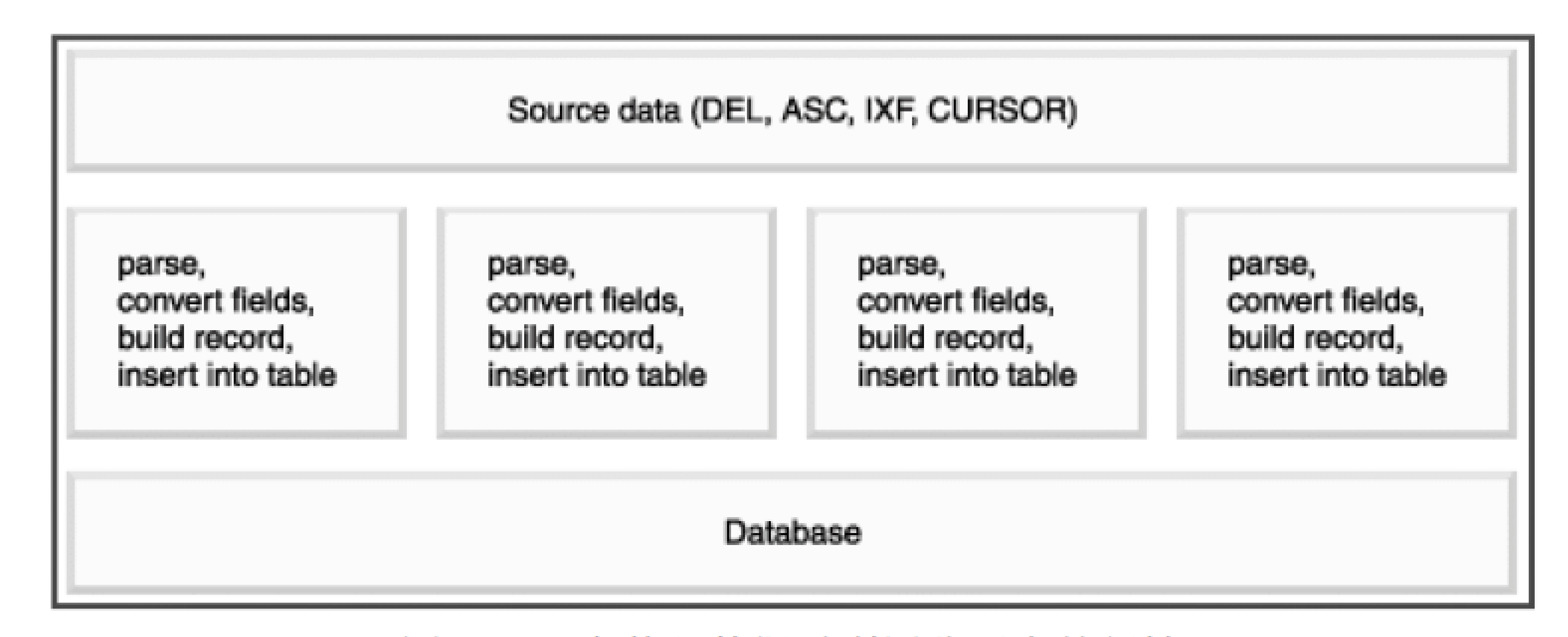


图 6-15　在装入数据时利用分区内并行性

2. 用于提高装入性能的选项

如果用户未指定 DISK_PARALLELISM、CPU_PARALLELISM 和 DATA BUFFER 参数的值，那么 LOAD 实用程序将尝试通过确定这些参数的最优值来获得最好的性能。根据实用程序堆大小及可用空间来进行优化。在尝试调整这些参数以满足特殊需要之前，考虑使用自主 DISK_PARALLELISM 和 CPU_PARALLELISM 设置。

以下是有关 LOAD 实用程序提供的各种选项所带来的性能影响的信息：

- ALLOW READ ACCESS

此选项允许您在进行装入操作时查询表。只能查看装入操作之前表中已存在的数据。如果还指定了 INDEXING MODE INCREMENTAL 选项，并且装入操作失败，那么后续装

入终止操作可能必须校正索引中的不一致。这需要涉及大量 I/O 的索引扫描。如果还对装入终止操作指定了 ALLOW READ ACCESS 选项，那么会将缓冲池用于 I/O。这个选项会间接地影响 LOAD 的时间，因为在线 LOAD 会延长装载时间。

- COPY YES 或 COPY NO

使用此参数来指定在装入操作期间是否创建输入数据的副本。仅当启用了前滚恢复时，COPY YES 才适用，并且由于装入操作期间会复制所有装入数据，因此使用此参数会降低装入性能。I/O 活动增加可能会导致 I/O 绑定系统上的装入时间增加。如果指定了多个设备或不同磁盘上的多个目录，那么可能会因为此操作而使性能受到影响。仅当启用了前滚恢复时，COPY NO 才适用，并且不会影响装入性能。但是，所有与已装入的表相关的表空间将处于“备份暂挂”状态，并且必须先备份这些表空间才能访问该表。

- CPU_PARALLELISM

借助此参数来使用分区内并行性(如果机器具有此功能)，从而大幅改进装入性能。该参数指定 LOAD 实用程序用于分析、转换、格式化数据记录的进程或线程的数目。允许的最大数目是 30。如果内存不足以支持指定值，那么实用程序将调整该值。如果未指定此参数，那么 LOAD 实用程序将根据系统上的 CPU 数目选择默认值。

只要满足下列条件，无论此参数的值如何，都将保留源数据中的记录顺序(请参阅图 6-16)：

◊ 未指定 ANYORDER 文件类型修饰符。

◊ 未指定 PARTITIONING_DBPARTNUMS 选项(并且将多个分区用于分区)。

◊ 如果表包括 LOB 或 LONG VARCHAR 数据，那么 CPU_PARALLELISM 将设置为 1。在这种情况下不支持并行性。

尽管此参数并未限制为只能供对称多处理器(SMP)硬件使用，但在非 SMP 环境中使用它在性能方面也没什么太大益处。



图 6-16　在装入操作期间使用分区内并行性时，将会保留源数据中的记录顺序

- DATA BUFFER

DATA BUFFER 参数指定分配给 LOAD 实用程序以用作缓冲区的内存总量(以 4KB 为单位)。建议此缓冲区在大小上等于若干扩展数据块。数据缓冲区将从实用程序堆分配。根据系统上可用的存储量，应考虑分配更多内存以供 DB2 实用程序使用。可相应修改数据库配置参数 util_heap_sz(实用程序堆大小)。util_heap_sz 的默认值为 5000 个 4KB 的页。因为

LOAD 实用程序只是使用实用程序堆内存的若干实用程序的其中一个，所以建议将此参数定义的不超过 50%的页供 LOAD 实用程序使用，并且将实用程序堆定义得足够大。

- DISK_PARALLELISM

DISK_PARALLELISM 参数指定 LOAD 实用程序用来将数据记录写至磁盘的进程或线程数。借助此参数在装入数据时使用可用容器，从而大幅改进装入性能。允许的最大数目是 CPU_PARALLELISM 值(LOAD 实用程序使用的实际量)的 4 倍或 50 中较大的数字。默认情况下，DISK_PARALLELISM 等于包含对其装入表的对象的所有表空间中的表空间容器的总和，但当此值超过允许的最大值时除外。

- SAVECOUNT

使用此参数来设置在装入操作的装入阶段建立一致点的时间间隔。为建立一致点而执行活动同步需要花一些时间。如果进行得太频繁，装入性能会大幅下降。如果要装入大量行，那么建议您指定较大的 SAVECOUNT 值(例如，在涉及 1 亿条记录的装入操作中指定值 10 000 000)。只要装入重新启动操作(restart)从装入阶段恢复，该操作就会从上一个一致点自动继续。

- STATISTICS USE PROFILE

收集表统计信息概要文件中指定的统计信息。即使装入操作本身的性能下降(特别是在指定 DETAILED INDEXES ALL 时)，与在完成装入操作后调用 RUNSTATS 实用程序相比，使用此参数来收集数据分发和索引统计信息更有效。为优化性能，应用程序需要尽可能最佳的数据分发和索引统计信息。一旦更新统计信息，应用程序就可以根据最新的统计信息使用新的表数据访问路径。可通过使用 BIND 命令重新绑定应用程序包来创建新的表访问路径。通过运行带有 SET PROFILE 选项的 RUNSTATS 命令来创建表统计信息概要文件。

将数据装入到大表中时，建议对 stat_heap_sz(统计信息堆大小)数据库配置参数指定较大的值。

- USE <tablespace-name>

如果正在执行 ALLOW READ ACCESS 装入并且建立索引的方式为 REBUILD，那么此参数允许在系统临时表空间中重建索引，并在装入操作的索引复制阶段将其复制回索引表空间。

默认情况下，将在原始索引所在的表空间中构建完全重建的索引(也称为影子索引)。因为原始索引和影子索引同时位于同一表空间中，所以这可能会导致资源问题。如果影子索引与原始索引是在同一表空间中构建的，那么影子索引将瞬时替换原始索引。但是，如果影子索引是在系统临时表空间中构建的，那么装入操作需要索引复制阶段，该阶段会将索引从系统临时表空间复制至索引表空间。复制阶段将涉及相当多的 I/O。如果其中任一表空间是 DMS 表空间，那么系统临时表空间的 I/O 可能不是顺序进行的。在索引复制阶

段将使用 DISK_PARALLELISM 选项指定的值。

3. 文件类型修饰符

文件类型修饰符是用 MODIFIED BY 子句指定的。下面是一些对性能会有影响的文件类型修饰符：

- ANYORDER

默认情况下，LOAD 实用程序将保留源数据的记录顺序。在 SMP 环境中进行装入时，要求并行处理之间保持同步以确保保留该顺序。

在 SMP 环境中，指定 ANYORDER 文件类型修饰符将指示 LOAD 实用程序不保留顺序，由于这样一来就不必执行保留该顺序所需的同步，因此将会提高性能。但是，如果要装入的数据进行了预先排序，那么 ANYORDER 可能会破坏预先排好的顺序，使得后续查询也无法受益于预先排序。

注意：

对于外界来源的数据在使用时，如果 CPU_PARALLELISM 为 1，那么 ANYORDER 文件类型修饰符不起作用，并且它与 SAVECOUNT 选项不兼容。

- BINARYNUMERICS、ZONEDDECIMAL 和 PACKEDDECIMAL

对于固定长度的非定界 ASCII(ASC)源数据，用二进制表示数字数据可能会提高装入时的性能。如果指定了 PACKEDDECIMAL 文件类型修饰符，那么 LOAD 实用程序会使用压缩十进制格式(每个字节占两位)表示十进制数据。如果指定了 ZONEDDECIMAL 文件类型修饰符，那么 LOAD 实用程序会使用分区十进制格式(每个字节占一位)表示十进制数据。对于所有其他数字类型，如果指定了 BINARYNUMERICS 文件类型修饰符，那么 LOAD 实用程序会使用二进制格式表示数据。

- FASTPARSE

使用时务必小心谨慎。如果知道要装入的数据有效，那么没必要让装入像对较可疑的数据那样执行那么多的语法检查。事实上，缩小语法检查的范围可以将装入性能提高大约 10%或 20%。这可以通过使用 FASTPARSE 文件类型修饰符来实现，该修饰符可以减少对 ASC 和 DEL 文件中用户提供的列值执行的数据检查。

- PAGEFREESPACE、INDEXFREESPACE 和 TOTALFREESPACE

随着时间的推移，表中插入和更新的数据不断增加，重组表和索引的需求也就更迫切。一种解决方案是使用 PAGEFREESPACE、INDEXFREESPACE 和 TOTALFREESPACE 来增

大用于表和索引的可用空间量。前两个修饰符优先于 PCTFREE 值，它们指定要作为可用空间保留的数据和索引页数的百分比，而 TOTALFREESPACE 指定要作为可用空间追加至表的总页数的百分比。

4. LOAD 性能总结

图 6-17 是使用上述命令选项和文件修饰符前后的性能比较。

文件格式IXF性能最好，建议多用IXF	使用的LOAD命令选型和Modified by文件修饰符对性能的影响。因为硬件环境不一样，读者应主要关心使用这些影响LOAD性能选项后的性能相对提高度	GB/h/CPU	Rows/Sec	Improvement
IXF	ANYORDER, DATA BUFFER 40000, CPU_PARALLELISM 5, DISK_PARALLELISM 8	50.8±0.2	127900±300	18%
ASC	ANYORDER, FASTPARSE, DATA BUFFER 40000, CPU_PARALLELISM 6, DISK_PARALLELISM 8	29.2±0.1	57300±200	36%
DEL	ANYORDER, FASTPARSE, DATA BUFFER 40000, CPU_PARALLELISM 6, DISK_PARALLELISM 8	28.0±0.1	54800±60	29%
CURSOR	ANYORDER, DATA BUFFER 40000, CPU_PARALLELISM 2, DISK_PARALLELISM 8, INTRA_PARALLEL ON, DFT_DEGREE 18	14.0±0.1	50200±300	14%

图 6-17　使用命令选项和文件修饰符前后的性能比较

从图 6-17 中我们可以看到，使用 IXF 格式的性能比 DEL 和 ASC 要高，所以建议大家在数据移动时最好使用 IXF 格式。同时因为硬件环境的差异，大家只需要关注使用这些选项的相对提高度，而不必关注特定的 LOAD 时间。

6.4.9　LOAD 失败恢复

DB2 将在装入处理期间创建临时二进制文件。这些文件用于装入崩溃恢复、装入终止操作、警告和错误消息以及运行时控制数据。装入临时文件将在装入操作完成而未发生任何错误时自动清除。临时文件将写至通过 LOAD 命令的 temp-pathname 参数指定的路径。默认路径为数据库目录的子目录。

临时文件路径在服务器上，并且由 DB2 实例以独占方式访问。因此，对 temp-pathname 参数指定的任何路径名限定都必须反映服务器(而不是客户机)的目录结构，并且 DB2 实例所有者对该路径必须具有读写许可权。图 6-18 显示了 LOAD 异常终止产生的临时文件。



图 6-18　LOAD 异常终止产生的临时文件

当 LOAD 正常结束时，临时文件自动删除。临时文件主要用于异常恢复。

注意：

写至此路径的临时文件在任何情况下都不能编辑修改。编辑修改这些临时文件将导致装入操作失败，并且会使数据库陷入危险状况。

解决导致装入操作失败的情况后，重新发出 LOAD 命令。确保指定的参数与原始命令中的参数完全相同，以便 LOAD 实用程序可以找到必需的临时文件终止该操作、重新装入表或重新启动装入操作。如果要禁止读访问，那么不必指定完全相同的参数。还可以将指定了 ALLOW READ ACCESS 选项的装入操作作为 ALLOW NO ACCESS 选项重新启动。

如果 LOAD 实用程序因为用户错误(例如，数据文件不存在或列名无效)而不能启动，那么操作将终止并让目标表处于正常状态。

装入操作开始时，目标表将处于“正在装入”状态。出现故障时，表状态将更改为“装入暂挂”。要使表脱离该状态，可以发出 LOAD TERMINATE 以回滚操作，发出 LOAD REPLACE 以重新装入整个表，或者发出 LOAD RESTART。

通常，在这种情况下，最好重新启动装入操作。由于 LOAD 实用程序是从装入操作最后成功到达的位置而不是从该操作的开头重新启动装入操作，因此这样做可以节省时间。操作重新启动的准确位置取决于在原始命令中指定的参数。如果指定了 SAVECOUNT 选项，并且上一个装入操作在装入阶段失败，那么装入操作将在它到达的最后一个一致点重新启动。否则，装入操作在成功到达的最后一个阶段(装入、构建或删除阶段)开始时重新启动。

如果要装入 XML 文档，那么该行为稍有不同。因为在装入 XML 数据时不支持 SAVECOUNT 选项，所以在装入阶段失败的装入操作将从操作的起始处重新启动。正如其他数据类型一样，如果在构建阶段装入失败，那么将在 REBUILD 方式下构建索引，因此会扫描该表以便从每一行获取所有索引键；但是，也必须扫描每个 XML 文档以获取索引键。扫描 XML 文档以查找索引键的这一过程要求对它们重新进行语法分析，这是成本高昂的操作。而且，诸如区域和路径索引之类的内部 XML 索引需要先重构，这也要求扫描 XDA 对象。

如果下列命令产生的装入操作失败：

```
LOAD FROM file_name OF file_type SAVECOUNT n
MESSAGES message_file  load_method INTO target_tablename
```

那么可以通过将指定的装入方法(load_method)替换为 RESTART 方法来重新启动该操作：

```
LOAD FROM file_name OF file_type  SAVECOUNT n
MESSAGES message_file  RESTART   INTO target_tablename
```

不能重新启动(restart)的失败装入

如果失败或中断的装入操作中使用的表处于“不可重新启动装入”状态，那么不能重新启动该操作。不能重新启动的原因如下：

- 在未成功地重新启动或终止的失败装入操作后执行了前滚操作
- 根据表处于“正在装入”或“装入暂挂”状态时创建的联机备份执行了复原操作

应该发出 LOAD TERMINATE 或 LOAD REPLACE 命令。

重新启动或终止 ALLOW READ ACCESS 装入操作

如果以 ALLOW READ ACCESS 方式进行的装入操作在装入(load)阶段被中断或取消，那么可以使用 ALLOW READ ACCESS 选项重新启动(restart)或终止(terminate)指定了 ALLOW READ ACCESS 选项的已中断或已取消的装入操作。它将在装入阶段重新启动。如果它在装入阶段以外的任何阶段(build、delete 和 index copy)被中断或取消，那么它将在构建(build)阶段重新启动。但是如果索引对象不可用或标记为无效，那么不允许 ALLOW READ ACCESS 方式的装入重新启动。如果原始装入操作在索引复制阶段被中断或取消，那么因为索引可能已损坏而不允许 ALLOW READ ACCESS 方式的重新启动操作。发出 LOAD TERMINATE 命令通常会导致已中断或已取消的装入操作以最短延迟回滚。但是，对指定了 ALLOW READ ACCESS 和 INDEXING MODE INCREMENTAL 的装入操作发出 LOAD TERMINATE 命令时，LOAD 实用程序扫描索引和校正任何不一致时会有延迟。此延迟的长度取决于索引的大小，并且无论是否对装入终止操作指定 ALLOW READ ACCESS 选项，都会发生延迟。如果原始装入操作在到达构建阶段前失败，那么不会发生延迟。

如果以 ALLOW NO ACCESS 方式执行装入操作，那么当原始装入操作到达构建阶段并且索引有效时，将在删除(delete)阶段发生重新启动操作。如果索引标记为无效，那么 LOAD 实用程序将从构建阶段重新启动装入操作。

6.4.10　LOAD 和 IMPORT 的比较

前面我们讲解了 IMPORT 和 LOAD，表 6-7 对 LOAD 和 IMPORT 进行了详细比较。

表 6-7　对 LOAD 和 IMPORT 的详细比较

IMPORT 实用程序	LOAD 实用程序
移动大量数据时速度较慢。本质是 SQL 操作，导入期间数据库写日志、检查约束，触发器，面向 row，所以速度比较慢	移动大量数据时，由于 LOAD 实用程序将格式化的页直接写入数据库，因此速度比 IMPORT 实用程序快
限制使用分区内并行性。只能通过在 ALLOW WRITE ACCESS 方式下并发调用 IMPORT 实用程序来实现分区内并行性	可使用分区内并行性。通常，这需要对称多处理器(SMP)环境。支持 CPU、DISK 和 FETCH 并行性
数据源是平面文件	数据源可以为平面文件、磁带和命名管道
不支持 FASTPARSE	支持 FASTPARSE，减少了对用户提供的数据进行的数据检查工作
索引不重建	索引重建，可以选择索引重建方式
能够创建 PC/IXF 格式的表、层次表和索引	表和索引必须存在
不支持导入到具体化查询表中	支持装入到具体化查询表中
ASC、DEL、WSF 和 IXF 文件格式	ASC、DEL、IXF 和 CURSOR
不支持 BINARYNUMERICS、PACKEDDECIMAL、ZONEDDECIMAL	支持 BINARYNUMERICS、PACKEDDECIMAL、ZONEDDECIMAL
无法覆盖定义为 GENERATED ALWAYS 的列	通过使用 GENERATEDOVERRIDE 和 INDENTITYOVERRIDE 文件类型修饰符，可以覆盖 GENERATED ALWAYS 列
支持导入到表、视图和昵称中	仅支持装入到表中
记录所有行	执行最少的记录
导入期间触发器工作	不支持触发器
如果导入操作被中断，并且指定了 COMMITCOUNT，那么该表可供使用，并且将包含最后一次落实(COMMIT)之前已装入的行。用户可以重新启动导入操作，也可以按原样接受　该表	如果装入操作被中断，并且指定了 SAVECOUNT，那么在重新启动装入操作、调用装入终止操作或者根据尝试执行装入操作前创建的备份映像复原表空间之前，该表将保持装入暂挂状态并且不可用
所需空间量大概等于最大索引大小加上 10%。此空间是从数据库中的临时表空间中获取的	所需空间量大概等于对该表定义的所有索引的大小之和，并且可能是此大小的两倍。此空间是从数据库中的临时空间中获取的

(续表)

IMPORT 实用程序	LOAD 实用程序
在导入操作期间将验证所有约束	LOAD 实用程序将检查唯一性并计算生成列值，但必须使用 SET INTEGRITY 来检查所有其他约束
在导入操作期间，将逐个地把键值插入到索引中	对键值进行排序，装入数据后构建索引
如果需要更新的统计信息，导入操作完成后必须运行 RUNSTATS 实用程序	如果正在替换表中的所有数据，那么可以在装入操作执行期间收集统计信息
支持 XML 导入	不支持 XML 导入
导入文件必须在调用 IMPORT 实用程序的客户机上	根据指定的选项，装入文件或管道可以在包含数据库的数据库分区上，也可以在所连接的调用 LOAD 实用程序的远程客户机上 注意：只能从服务器端读取 LOB 和 XML 数据
不需要备份映像。由于 IMPORT 实用程序使用 SQL 插入，因此将记录活动，发生故障时不需要备份就可以恢复这些操作	在装入操作执行期间，指定 COPY YES，可以自动创建表空间备份映像

6.5 数据移动的性能问题

如果我们希望提高数据移动的速度，那么对于 IMPORT 和 EXPORT 来说，它们的本质是执行 SQL 语句，所以设置大的缓冲池、util_heap_sz 和排序堆(sortheap)会提高它们的速度。对于 LOAD 来说，我们在 6.4.8 节已经讲解。这里我们再次回顾一下。

compound x

指定 DB2 IMPORT 实用工具每次插入一块(x 行)数据，而非每次插入一行。这可以导致性能的提高。x 的值可以是 1 至 100 之间的任意整数(包含 1 和 100 在内)。IMPORT 实用工具使用非原子复合 SQL 来插入数据：不管是否有错，都会尝试所有的插入。使用该选项大概可以有 30%左右的性能提升。

```
    create table emptemp like employee;
    export to empdata.ixf of ixf messages export.msg select * from employee;
    import from empdata.ixf of ixf modified by compound=100 messages import.msg
insert into emptemp;
```

data buffer x

指定在执行 LOAD 的时候使用的数据缓冲的大小，单位为 4KB。在加载大量数据的时

候将此值设置较大能够极大地提高数据加载的性能。这部分内存将会从数据库的由 util_heap_sz 指定的内存中分配。所以如果需要较大的 data buffer，那么需要同时增加数据库参数 util_heap_sz 的大小。如果不指定此选项，LOAD 会在启动的时候自动地以空闲的 util_heap_sz 中的一定比例来分配内存。

使用方法如下：

```
load from test.del of del insert into tbname data buffer 2000
```

CPU_PARALLELISM　x

指定在运行 LOAD 时，为了解析、转换和格式化而创建的进程数或线程数，从而可以利用分区内并行性来提高性能。如果加载的数据是经过排序的，那么此选项特别有用。如果此参数为 0 或不指定此参数，那么 LOAD 会自动地设置一个值，比如当前可用的 CPU 的数量。

语法如下：

```
load from test.del of del insert into tbname CPU_PARALLELISM 10
```

DISK_ PARALLELISM x

指定在运行 LOAD 时，为了将数据写入表空间的容器中而创建的进程数或线程数。如果目标表空间中存在多个容器，那么此选项可以极大地提高 I/O 性能。

语法如下：

```
load from test.del of del insert into tbname DISK_PARALLELISM 10
```

除了 IMPORT、EXPORT 和 LOAD 工具的文件修饰符外，数据库层面的 bufferpool、util_heap_sz 和 sortheap 的大小对数据移动也至关重要。

6.6　db2move 和 db2look

DB2 有一对非常有用的工具，可以帮助您实现这种跨平台的数据移动。db2move 是用于在 DB2 数据库之间移动大量表的数据移动工具。db2move 利用 DB2 的数据移动工具(EXPORT、IMPORT、LOAD 和 COPY)来移动数据库表。然而，由于数据库的内容远远不止用户表，因此您需要使用其他方法在不同的数据库之间迁移其他数据库对象，例如约束、触发器、索引、序列、表空间、缓冲池等。这就是 db2look 工具出现的原因。使用这个工具，您可以在源数据库中捕获到定义这些对象的数据定义语言(DDL)，并在目标数据库中使用这些数据定义语言重新创建这些对象。

6.6.1 数据库移动工具——db2move

db2move 工具将一组用户表从系统编目表中提取出来，并将每个表以 PC/IXF 格式导出。然后，PC/IXF 文件可以被导入或装载到另一个 DB2 数据库中。这些 PC/IXF 文件既可以被导入或装载到同种系统上的其他本地 DB2 数据库中，也可以被传递到其他操作系统平台上，并导入或装载到这种平台上的 DB2 数据库中；db2move 工具在导出操作中生成的文件可以用作后来这些导入或装载操作的输入文件(参见表 6-8)。要使 db2move 操作成功执行，所使用的用户 ID 必须具有底层 DB2 数据移动工具所需要的适当授权。在调用 db2move 命令之前，并不需要数据库连接；该工具会为您建立数据库连接。

这个命令中支持的动作有 EXPORT、IMPORT、LOAD 和 COPY。db2move 的语法很简单。

db2move 命令的基本语法如下所示：

```
db2move <database-name> <action> [<option> <value>]
```

首先，您必须指定数据库名(想要移动的表所在的数据库)和要执行的操作(EXPORT、IMPORT、LOAD 和 COPY)。然后指定一个选项来定义操作的范围。例如，可以将操作限制在特定的表(-tn)、表空间(-ts)、表创建者(-tc)或模式名(-sn)范围内。指定表、表空间或表的创建者的子集只对 EXPORT 操作有效。如果指定多个值，就必须使用逗号将它们分隔开；值列表项之间不允许有空格。可以指定的项最多为 10 个。另外，也可以指定-tf 选项，此时要使用文件名作为参数，其中列出了要导出的表名；在该文件中，每行只能列出完整的表名。您还可以指定以下内容：

```
-io import-option
```

指定 DB2 的 IMPORT 工具可以运行的一种模式。有效的选项有：CREATE、INSERT、INSERT_UPDATE、REPLACE 和 REPLACE_CREATE。默认值为 REPLACE_CREATE。

```
-lo load-option
```

指定 DB2 的 LOAD 工具可以运行的一种模式。有效的选项有：INSERT 和 REPLACE。默认值为 INSERT。

```
-l lobpaths
```

指定要创建或查找的 LOB 文件的位置。必须指定一个或多个绝对路径名。如果指定了多个绝对路径，就必须使用逗号将它们分隔开；值之间不允许有空格。默认值是当前目录。

```
-u userid
```

指定一个用户 ID，db2move 工具可以使用这个用户 ID 登录到远程系统上。

```
-p password
```

指定对该用户进行认证的密码；db2move 工具需要使用有效的用户 ID 和密码登录到远程系统上。

下面是一些例子。db2move 命令用指定的用户 ID 和密码以 REPLACE 模式导入 sample 数据库中的所有表：

```
db2move sample IMPORT -io REPLACE -u userid -p password
```

下面的命令以 REPLACE 模式装载 db2admin 和 db2inst1 这两个模式下的所有表：

```
db2move sample LOAD -sn db2admin,db2inst1 -lo REPLACE
```

db2move 在执行期间会在所在目录下生成一些文件，这些文件如表 6-8 所示。

表 6-8　db2move 在 EXPORT、IMPORT 和 LOAD 操作过程中需要或生成的文件

EXPORT	IMPORT		LOAD	
无输入文件，只有输出文件	输入文件	输出文件	输入文件	输出文件
EXPORT.out ① db2move.lst ② tab n.ixf ③ tab n.msg ④ tab na.nnn ⑤ system.msg ⑥		IMPORT.out ①		LOAD.out ①
	db2move.lst ②		db2move.lst ①	
	tab n.ixf ③		tab n.ixf ③	
		tab n.msg ④		tab n.msg ④
	tab na.nnn ⑤		tab na.nnn ⑤	

注：

① 包含对全部操作的摘要(ASCII 格式)。

② 包含源数据库中源表名的列表、对应的 PC/IXF 文件名和消息文件名(ASCII 格式)。

③ 包含从用户表中导出的数据，使用 n 标识(二进制格式)。

④ 包含对用户表请求操作的消息，使用 n 标识(ASCII 格式)。

⑤ 包含对用户表导出的大对象(LOB)数据，使用 n 标识。该文件的扩展名是数字，范围从 001 到 999；其中 a 是字符。这些 LOB 文件只有在被导出的表中包含 LOB 数据时才会被创建，保存在 LOB 路径目录中(二进制格式)。

⑥ 包含系统消息，只有在执行 EXPORT 操作并且已经指定好 LOB 路径时才会被创建(ASCII 格式)。

6.6.2 DB2 DDL 提取工具——db2look

db2look 工具提取的 DDL 语句，可以用于在其他系统上重建数据库对象。在调用 db2look 命令之前，不需要提前建立数据库连接；这个工具会为您建立数据库连接。

db2look 命令的基本语法如下所示：

```
db2look -d <database-name> [<option1> <option2 <option>]
```

首先，您必须指定想要描述的对象所在的数据库名。然后指定一个或多个选项(可以是任意顺序)来定义提取的范围，包括：

- **-e** 提取数据库对象的 DDL 语句，例如表、视图、自动摘要表、索引、触发器、序列、主键、引用、检查约束、用户定义函数和过程。
- **-a** 提取用户创建的所有对象的 DDL 语句。如果这个选项与 -e 选项一起指定，那么就要对数据库中的所有对象进行处理。
- **-z schema-name** 将输出限制为具有指定模式名的对象。
- **-t table-name** 将输出限制在一个或多个(最多 30 个)指定的表中。表名必须使用空格字符分隔开。
- **-m** 生成需要的 UPDATE 语句，对表、列和索引的统计信息进行复制。
- **-l** 为用户定义的表空间、数据库分区组和缓冲池生成 DDL 语句。
- **-x** 生成对数据库对象进行授权或回收权限的 DDL 语句。
- **-td delimiter** 指定 db2look 工具使用的分隔符；默认为分号(;)。
- **-o file-name** 将输出结果写入文件。如果没有指定该选项，就将输出结果写入标准输出设备。
- **-i userid** 指定用户 ID，db2look 工具需要使用它登录到远程系统上。
- **-w password** 指定对用户进行认证的密码；db2look 工具需要使用有效的用户 ID 和密码登录到远程系统上。

6.6.3 利用 db2move 和 db2look 移动数据的案例

现在让我们来看一个实际的例子。在一个正在运行 DB2 V8.2.4 数据库的 AIX V5.3 的系统上有一个 PROD 数据库。PROD 有 5 个表：MOVIE、ACTOR、APPEARS_IN、DIRECTOR 和 DIRECTS，每个表中都包含数据。其中有些表上定义了主键，ACTOR 表上定义了如下检查约束：ACTOR_AGE 约束要求 ACT_YR_OF_BIRTH 列的值必须在 2004 年之前。

注意：

IXF 格式虽然包含表和索引的定义，但是不包含检查约束的定义。

我们要将 PROD 数据库复制到同样也在运行 DB2 V8.2.4 的 Windows XP 系统上。我们采取的策略是首先使用 db2move 将所有表的数据导出为 PC/IXF 文件，然后使用 db2look 为现有的数据库对象捕获 DDL 语句，包括 ACTOR_AGE 检查约束，它不会包含在 PC/IXF 文件中。接下来使用 FTP 将从这些工具中得到的输出文件传递到 Windows 系统上，在 Windows 系统上重新创建数据库以及其中的对象，最后运行 db2move 工具来装载 PC/IXF 文件中包含的数据。步骤如下：

(1) 在 AIX 上，运行 db2move，导出 PROD 数据库中所有用户表中的数据：

```
$/home/prodinst>db2move PROD export
*****  DB2MOVE  *****
Action:      EXPORT
Connecting to database PROD ... successful!  Server: DB2 Common Server V8.2.4
Binding package automatically ...
Bind file: /home/prodinst/sqllib/bnd/db2move.bnd
Bind was successful!
EXPORT:     44 rows from table "PRODINST  "."DIRECTS"
EXPORT:     76 rows from table "PRODINST  "."APPEARS_IN"
EXPORT:     40 rows from table "PRODINST  "."MOVIE"
EXPORT:     70 rows from table "PRODINST  "."ACTOR"
EXPORT:     42 rows from table "PRODINST  "."DIRECTOR"
Disconnecting from database ... successful!
End time:  Fri Mar 12 23:04:18 2008
```

(2) 在 AIX 上，运行 db2look，为 PROD 数据库中的所有对象捕获 DDL 语句，并将输出结果写入到名为 db2look.sql 的文件中：

```
$/home/prodinst>db2look -d PROD -e -a -o db2look.sql
-- Generate statistics for all creators
-- Creating DDL for table(s)
-- Output is sent to file: db2look.sql
-- Binding package automatically ...
-- Bind is successful
```

(3) 在 Windows 上，使用 FTP 登录到 AIX 系统上，并下载最后的 db2move 操作所需要的输入文件。请确保使用二进制模式传输 PC/IXF 文件，使用 ASCII 模式传输 db2move.lst 文件和 db2look.sql 文件：

```
ftp> prompt
Interactive mode Off .
ftp> bin
200 Type set to I.
ftp> mget *.ixf
200 Type set to I.
200 PORT command successful.
150 Opening data connection for tab1.ixf (4513 bytes).
226 Transfer complete.
ftp: 4513 bytes received in 0.13Seconds 34.45Kbytes/sec.
 ...
200 PORT command successful.
150 Opening data connection for tab5.ixf (6289 bytes).
226 Transfer complete.
ftp: 6289 bytes received in 0.12Seconds 52.41Kbytes/sec.
ftp> asc
200 Type set to A; form set to N.
ftp> get db2move.lst
200 PORT command successful.
150 Opening data connection for db2move.lst (205 bytes).
226 Transfer complete.
ftp: 210 bytes received in 0.01Seconds 21.00Kbytes/sec.
ftp> get db2look.sql
200 PORT command successful.
150 Opening data connection for db2look.sql (2754 bytes).
226 Transfer complete.
ftp: 2876 bytes received in 0.15Seconds 19.17Kbytes/sec.
ftp> bye
221 Goodbye.
```

(4) 在 Windows XP 上，创建 PROD 数据库，然后运行 db2look 工具生成的脚本以创建数据库对象，包括用户表和检查约束 ACTOR_AGE。运行 db2move，将数据从 PC/IXF 文件装载到 PROD 数据库的所有用户表中：

```
C:\Db2move>db2 create db PROD
C:\Db2move>db2 -tvf db2look.sql
C:\Db2move>db2move PROD load
*****  DB2MOVE  *****
Action:      LOAD
Connecting to database PROD ... successful!  Server: DB2 Common Server V8.2.4
Binding package automatically ...
Bind file: C:\Program files\IBM\SQLLIB\BND\DB2MOVE.BND
Bind was successful!
```

```
* LOAD:  table "PRODINST  "."DIRECTS"
  -Rows read:  44  -Loaded:  44  -Rejected   0  -Deleted   0  -Committed  44
* LOAD:  table "PRODINST  "."APPEARS_IN"
  -Rows read:  76  -Loaded:  76  -Rejected   0  -Deleted   0  -Committed  76
* LOAD:  table "PRODINST  "."MOVIE"
  -Rows read:  40  -Loaded:  40  -Rejected   0  -Deleted   0  -Committed  40
* LOAD:  table "PRODINST  "."ACTOR"
  -Rows read:  70  -Loaded:  70  -Rejected   0  -Deleted   0  -Committed  70
* LOAD:  table "PRODINST  "."DIRECTOR"
  -Rows read:  42  -Loaded:  42  -Rejected   0  -Deleted   0  -Committed  42
Disconnecting from database ... successful!
End time:  Sat Mar 13 21:01:25 2008
```

(5) 在 Windows XP 上，验证所复制的 PROD 数据库是否完好无损，ACTOR_AGE 检查约束是否可以正常工作：

```
C:\Db2move>db2 connect to PROD
C:\Db2move>db2 select * from movie fetch first 5 rows only
MOVIE ID TITLE                          YR RELEASED
-------- ------------------------------ -----------
23154    Carousel                             1956
44524    El Cid                               1961
78456    Giant                                1956
45692    African Queen                        1951
67845    Casablanca                           1942
  5 record(s) selected.
C:\Db2move>db2 select * from actor fetch first 5 rows only
ACTOR ID ACTOR NAME          ACT YR OF BIRTH
-------- -------------------- ---------------
SQL0668N  Operation not allowed for reason code "1" on table "PRODINST.ACTOR".
SQLSTATE=57016
C:\Db2move>db2 set integrity for actor immediate checked
C:\Db2move>db2 insert into actor values ('58825','Naomi Watts',2009)
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL0545N  The requested operation is not allowed because a row does not
satisfy the check constraint "PRODINST.ACTOR.ACTOR_AGE".  SQLSTATE=23513
C:\Db2move>db2 set integrity for prodinst.actor immediate checked
C:\Db2move>db2 insert into actor values ('58825','Naomi Watts',1968)
C:\Db2move>db2 "select * from actor where act_yr_of_birth > 1960"
ACTOR ID ACTOR NAME          ACT YR OF BIRTH
-------- -------------------- ---------------
```

```
    58825    Naomi Watts                    1968
    46739    Elaine Cassidy                 1980
    44333    Adam Baldwin                   1962
    44445    Tom Cruise                     1962
      4 record(s) selected.
```

ACTOR 表最初处于一种检查挂起状态(由于检查约束的原因)，需要执行 db2 set integrity for prodinst.actor immediate checked 语句将其转换成正常状态。然而，后续的插入操作会失败，因为年份值违反了检查约束。最后我们使用一个有效的年份值，就可以成功插入了。

6.6.4 带 COPY 操作的 db2move 实用程序

db2move 命令中支持的动作有 EXPORT、IMPORT、LOAD 和 COPY。EXPORT、IMPORT 和 LOAD 这几个动作的行为与前面描述的完全相同。您可能不熟悉的唯一动作就是 COPY，它将一个或多个模式中的表复制到目标数据库中。在 COPY 动作中，可以用-sn 选项指定一个或多个模式。只有具有-sn 选项中指定的模式名的表才被复制(通过导出)。带 COPY 操作的 db2move 实用程序的语法如下：

```
>>-db2move--dbname—COPY----+----------------------+-+----->< 
                           +- -sn--schema-names-----+
                           +- -tn--table-names------+
                           +- -tf--filename---------+
                           +- -co--copy-option------+
                           +- -u--userid------------+
                           +- -p--password----------+
```

在使用带 COPY 操作的 db2move 实用程序时，可以在-co 后面使用以下选项：

- TARGET_DB <db name> [USER <userid> USING <password>]

允许用户指定目标数据库的名称以及用户名和密码(可以使用-p 和-u 选项指定源数据库的用户名和密码)。USER 或 USING 子句是可选的。如果 USER 指定用户 ID，那么必须在 USING 子句中提供密码；如果没有指定密码，那么 db2move 会提示输入密码。出现这个提示是由于后面要讨论的安全因素。TARGET_DB 是 OPY 操作的必要选项。TARGET_DB 不能与源数据库相同。COPY 操作要求至少输入一个模式(-sn)或一个表(-tn 或-tf)。

如果要指定多个模式名，那么使用逗号将它们隔开，这里不允许使用空格。请参考下面的例子：

```
db2move sample COPY -sn db2inst1, prodschema -co TARGET_DB acctdb  USER nxz
USING nxz DDL_AND_LOAD
```

上面的 db2move 命令复制 db2inst1 和 prodschema 模式下受支持的对象。后面跟着的-co

选项使这个命令更加有趣。TARGET_DB 选项指定这些模式将被复制到的目标数据库。当指定了 COPY 动作时，这个选项是强制性的。此外，目标数据库必须不同于源数据库。当连接到目标数据库时，可以通过 USER 和 USING 选项提供用户名和密码。

- MODE

DDL_AND_LOAD：从源模式创建支持的所有对象，并用源表数据填充表。这是默认选项。

DDL_ONLY：从源模式创建支持的所有对象，但是不重新填充表。

LOAD_ONLY：将指定的所有表从源数据库装载到目标数据库中。这些表必须已经在目标数据库中存在。

- SCHEMA_MAP

允许用户在向目标数据库进行复制时对模式重新命名。需要提供源-目标模式映射列表，映射由括号包围，由逗号分隔，例如 schema_map((s1，t1)，(s2，t2))。这意味着把模式 s1 中的对象复制到目标数据库的模式 t1 中，把模式 s2 中的对象复制到目标数据库的模式 t2 中。默认的目标模式名与源模式名相同，这也是推荐的做法。这是因为 DM2MOVE 不会尝试修改对象体中任何限定对象的模式。因此，如果对象体中有限定对象，那么不同的目标模式名就可能导致问题。

在使用 SCHEMA_MAP 选项时要特别小心。只有对象本身的模式被重命名，而对象体中的对象仍保持不变。例如：

```
CREATE VIEW FOO.v1 AS 'SELECT c1 FROM FOO.T1'
```

将模式从 FOO 重命名为 BAR 将导致：

```
CREATE VIEW BAR.v1 AS 'SELECT c1 FROM FOO.T1'
```

如果 FOO.T1 没有定义，那么目标数据库中就不能成功地创建 BAR.v1。

- NONRECOVERABLE

这个选项允许用户改变 COPY-NO 装载操作的默认行为。在采用默认行为时，用户必须对进行装载的每个表空间进行备份。如果指定 NONRECOVERABLE 关键字，那么用户不必马上对表空间进行备份。但是，强烈建议尽快进行备份，从而确保新创建的表可以被正确地恢复。

- OWNER

允许用户在成功地 COPY 操作之后修改在目标模式中创建的每个新对象的所有者。目标对象默认的所有者是进行连接的用户。如果指定这个选项，那么所有者就改为新的所有者。

- TABLESPACE_MAP

用户可以指定在复制期间使用的表空间名映射，而不使用来自源数据库的表空间。这里应该提供表空间映射数组，映射由括号包围，例如 tablespace_map((TS1，TS2)，(TS3，TS4))。这意味着把表空间 TS1 中的所有对象复制到目标数据库的表空间 TS2 中，把表空间 TS3 中的对象复制到目标数据库的表空间 TS4 中。如果是((T1，T2)，(T2，T3))，那么源数据库 T1 中的所有对象在目标数据库的 T2 中重新创建，源数据库 T2 中的所有对象在目标数据库的 T3 中重新创建。默认情况下，使用与源数据库相同的表空间名，在这种情况下，不必提供这个表空间的映射。如果指定的表空间不存在，那么使用这个表空间的对象复制操作会失败，这一情况会记录在错误文件中。

用户还可以使用 SYS_ANY 关键字，这表示应该使用默认的表空间选择算法来选择表空间。在这种情况下，db2move 可以选择任何可用的表空间作为目标表空间。SYS_ANY 关键字可以用于所有表空间，例如 tablespace_map SYS_ANY。另外，用户可以为某些表空间指定特定的映射，而对其他表空间使用默认的表空间选择算法。例如 tablespace_map ((TS1，TS2)，(TS3，TS4)，SYS_ANY)，这表示表空间 TS1 映射为 TS2，TS3 映射为 TS4，而其他表空间将使用默认的表空间目标。使用 SYS_ANY 关键字是因为表空间名称不可能以“SYS”开头。

让我们来看一个综合性的例子。

```
    db2move sample COPY -sn db2inst1,prodschema    -co TARGET_DB acctdb USER nxz
USING nxzpwasswd LOAD_ONLY    SCHEMA_MAP
((db2inst1,db2inst2),(prodschema,devschema))
        TABLESPACE_MAP SYS_ANY    NONRECOVERABLE
```

这个命令将 db2inst1 和 prodschema 中受支持的对象从 sample 数据库复制到 acctdb 数据库。用户名 nxz 和密码 nxzpasswd 用于连接到 acctdb。目标表已经存在于 acctdb 中，这些表将被重新填充。db2inst1 和 prodschema 模式下的所有对象现在分别在 db2inst2 和 devschema 模式下。最后，不使用 sample 数据库中定义的表空间名称，而是使用 acctdb 中默认的表空间。

NONRECOVERABLE 选项允许用户在复制完成之后立即使用装载的目标表空间。这里不要求备份表空间，但是强烈建议在早期方便的时候做备份。

使用 db2move COPY 选项案例

在下面这个示例中，PROD 模式中的数据库修改成功地通过了测试，下面要把新的提交版本复制到新数据库中并命名为 DEV 模式。

使用示例：

```
db2move BANKDEV COPY -sn PROD -co TARGET_DB BANKSHIP USER  nxz USING nxzpasswd
MODE DDL_ONLY SCHEMA_MAP ((PROD,DEV))
```

只将模式 PROD 中的对象从源数据库 BANKDEV 复制到目标数据库 BANKSHIP 的模式 DEV 中：

```
db2move BANKDEV COPY -sn PROD -co TARGET_DB BANKSHIP USER  nxz USING nxzpasswd
SCHEMA_MAP ((PROD,DEV))
```

将模式 PROD 中的对象和包含的数据从源数据库 BANKDEV 复制到目标数据库 BANKSHIP 的模式 DEV 中：

```
db2move BANKDEV COPY -sn PROD -co TARGET_DB BANKSHIP USER USER nxz USING
nxzpasswd  SCHEMA_MAP ((PROD,DEV)) OWNER V9_ADMIN  TABLESPACE_MAP
((USERSPACE1,V9_USERSPACE1),(USERSPACE2,V9_USERSPACE2),SYS_ANY)
```

将模式 PROD 中的对象和包含的数据复制到模式 DE 中，并指定新的对象所有者 V9_ADMIN。将表空间 USERSPACE1 中的对象复制到表空间 V9_USERSPACE1 中。将表空间 USERSPACE2 中的对象复制到表空间 V9_USERSPACE2 中。其他表空间使用默认的表空间选择算法。

带 COPY 操作的 db2move 实用程序生成的文件如下：

- COPYSCHEMA.timestamp.msg：db2move COPY 操作产生的消息。
- COPYSCHEMA.timestamp.err：db2move COPY 操作产生的错误(只在发生错误的情况下生成这个文件)。
- LOADTABLE.timestamp.msg：db2move COPY 操作中由 LOAD 操作产生的消息(MODE DDL_AND_LOAD 和 LOAD_ONLY)。
- LOADTABLE.timestamp.err：db2move COPY 操作中由 LOAD 操作产生的错误(只在发生错误的情况下生成这个文件)。

例 6-17 成功执行带 COPY 操作的 db2move 实用程序后产生的消息：

```
Application code page not determined, using ANSI codepage 1386
*****  DB2MOVE  *****
Action:  COPY
Start time:  Mon Jun 12 17:46:39 2008
All schema names matching:  PROD;
Connecting to database BANKDEV ... successful!  Server : DB2 Common Server V9.5.0
Copy schema PROD to DEV on the target database BANKSHIP
Create DMT :  "SYSTOOLS"."DMT_448d83d5c76c"
```

```
db2move finished successfully
Files generated:
-----------------
COPYSCHEMA.20080612174639.msg
LOADTABLE.20080612174639.MSG
Please delete these files when they are no longer needed.
End time:  Mon Jun 12 17:46:43 2008
```

例 6-18 不成功地执行带 COPY 操作的 db2move 实用程序后产生的消息：

```
Application code page not determined, using ANSI codepage 1386
*****  DB2MOVE  *****
Action:  COPY
Start time:  Mon Jun 12 17:04:57 2008
All schema names matching:  PROD;
Connecting to database BANKDEV ... successful!  Server : DB2 Common Server V9.0.0
Copy schema PROD to DEV on the target database BANKSHIP
Create DMT :  "SYSTOOLS"."DMT 448d829c49bdd"
Rolled back all changes from the create phase (debuginfo:140).
db2move failed with -1 (debuginfo:220).
Files generated:
-----------------
COPYSCHEMA.20080612170457.msg
COPYSCHEMA.20080612170457.ERR
Please delete these files when they are no longer needed.
**Error occurred -1
End time:  Mon Jun 12 17:05:01 2008
Content of file COPYSCHEMA.20080612170457.ERR:
1 Schema        : PROD  .TEST
  Type          : TABLE
  Error Msg     : [IBM][CLI Driver][DB2/NT] SQL0204N
  "USERSPACE2" is an undefined name. SQLSTATE=42704
  DDL           :
 CREATE TABLE "DEV  "."TEST"  (  "COL1" INTEGER )    IN "USERSPACE2"
```

db2move 使用说明和限制

- db2move 实用程序尝试复制所有允许的模式对象，但是以下类型的对象除外：
  - ◊ 表层次结构
  - ◊ Java 例程存档(JARS)
  - ◊ 昵称(nickname)
  - ◊ 应用程序包

◊ 视图层次结构
◊ 对象特权(创建的所有新对象都具有默认的授权)
◊ 统计信息(新对象不包含统计信息)
◊ 索引扩展(与用户定义的结构化类型相关)
◊ 用户定义的结构化类型及其转换函数

- 在复制处理期间修改源模式中的表可能会导致在复制操作之后目标模式中的数据不相同。
- 对于不与模式相关的对象来说(比如表空间和事件监视器)，在模式复制操作期间不进行处理。
- 在对复制的(replicated)表进行复制时，表的新副本没有启用订阅。表只作为常规表重新创建。
- 如果源数据库和目标数据库不在同一实例中，那么必须对源数据库进行编目。
- 运行多个 db2move 命令将模式从一个数据库复制到另一个数据库会导致死锁。每次应该只执行一个 db2move 命令。

6.7　本章小结

本章我们讲解了 IMPORT、EXPORT、LOAD 和 db2move 这几种数据移动工具。其实 IMPORT 和 EXPORT 非常简单。LOAD 相对来说比较复杂，所以本章我们用了很大的篇幅来讲解 LOAD。在本章的最后，我们给大家讲解了 db2move 及其应用案例。大家应灵活掌握这些工具以便根据自己的需要选用最合适的工具来移动数据。



数据库备份与恢复

在以 IT 架构为基础的现代企业环境中，数据是企业最宝贵的资源。数据的丢失往往意味着难以计量和弥补的损失。而在数据库技术中，保证数据不丢失的最后一道屏障往往是数据库的备份。从这一角度出发，对数据进行备份与恢复，是数据库管理系统最重要的功能之一。是否具有灵活、健全的数据备份与恢复机制也是当前衡量关系数据库系统是否稳定、是否安全的重要指标之一。

为了更好地保护客户的关键数据，DB2 数据库提供了健全的备份、恢复功能，以方便用户制订并实施满足自己业务需求的数据保护策略。

本章主要讲解如下内容：

- 恢复的概念
- DB2 日志
- 数据库和表空间的备份
- 数据库和表空间的恢复
- 数据库和表空间的前滚
- RECOVER 数据库实用程序
- 数据库重建
- 监控备份、恢复和复原
- 优化备份、恢复和复原性能

7.1 恢复的概念

备份与恢复的概念

数据库的备份是数据库的副本以及一些控制信息，在出现故障的情况下，可以随时用

来进行恢复。数据库备份最小化了数据丢失，能够让你使用恢复过程，从备份副本中重新构造备份时的数据库。有多种情形会导致需要恢复数据库。下面列出了常见的需要使用数据库备份来恢复数据库中数据的主要场景：

- 程序逻辑错误

由于设计、编码或其他原因导致应用程序中存在逻辑错误，从而导致对数据库中的数据执行了不应该发生的修改、删除等操作，为了将这些错误数据修改恢复到正确的状态，就很有可能需要使用数据库的备份进行恢复。出现这种错误可能有许多不同的原因，但大多数情形与应用的开发和设计有关。

- 用户错误

对于数据库应用程序，有效但具破坏性的语句(比如删除整个工资表中的记录，误删除重要表，或者意外删除整个数据库)可能导致长时间停机。要避免这些错误，需要更多的正确培训和指导。数据定义语言(DDL)语句不能回滚。因此，如果用户的错误涉及 DDL，那么将需要采取正确的措施来恢复数据库。

例如，如果错误删除(drop)了表，那么作为 DBA 有两种选择。可以使用备份时刻的副本，并将它前滚至删除表之前的某个时间点(时间点恢复)；也可以通过 EXPORT 实用程序恢复逻辑备份。这两种选择都可能潜在地导致数据丢失。当用户删除 DB2 中的表时，可以执行数据库级的时间点恢复，前滚至删除表之前的时间点；或者最好使用表空间级的前滚操作，这样一来，不必让整个数据库停止服务，用户仍可以访问其他表空间中的数据。

- 数据库实例崩溃

数据库实例失败通常是由操作系统崩溃、停电或数据库本身的异常引起的。在 DB2 中，当数据库管理器和内存结构由于电源故障、磁盘损坏或网络故障而不能正常工作时，需要通过崩溃恢复将 DB2 恢复到一致可用的状态。

- 存储介质故障

当有人无意从文件系统删除了数据库文件时或由于存储故障导致数据库的数据文件不可用时，整个数据库都会处于一种不可用的状态。

还有另外一种情形，就是数据库中出现 bad page 错误，导致数据库无法使用，此种情形也有可能需要使用备份才能恢复数据库中的数据。

- 灾难

放置系统的设施遭到火灾、洪水、地震或其他类似灾难的毁坏。

当以上列举的情形发生时，很有可能就需要采用使用数据库的备份来恢复数据库，并使用日志前滚至指定的时间。你永远也无法知道系统何时会碰到灾难或故障。因此最好早作准备，不但要防止数据受到外部因素的影响，也要防止内部用户有意或无意中用不正确的信息破坏数据库。请考虑下列问题：你有备份自己的数据库吗？你能够恢复执行到最后

一秒的所有事务吗？ 如果答案是否定的，那么需要尽快行动起来，将上述问题的答案变成肯定的。否则后果真的会很严重。

笔者曾不止一次在不同单位遇到上述故障情形，而且由于没有数据库备份，使得数据库数据的恢复变更异常艰难甚至于不可能完成，即使最终能够恢复成功，但这一过程所需要的成本往往相当巨大，一般会远远超过对数据库执行备份所消耗的成本，考虑到不可恢复带来的更严重后果，还是请尽快行动起来，将所有的数据库都进行适当的备份，以保万一。

为了尽量减少数据的丢失，需要针对不同的项目制定合适的备份恢复策略，并确保备份恢复策略的可行性，而且还需要周期性地对恢复的方法进行演练以不断进行验证。

备份恢复策略

为了制订合适的备份恢复策略，应该问一问自己下面这些问题：

- 数据可以从另一个地方装载吗？
- 你能承受多少数据的丢失？
- 你想要花多少时间才能恢复数据库？
- 你有什么可用的资源来存储备份和日志文件？

事务

在讲解恢复类型之前，我们首先要了解事务的概念，事务、交易和工作单元(UOW)其实都是一个概念。关系型数据库为了保证数据库的可恢复性和一致性引入了事务这个概念。事务具有原子性、一致性、隔离性和永久性这几个特性。

- 原子性：事务的原子性指的是，事务中包含的 SQL 操作作为数据库的逻辑工作单元(UOW)，它所做的对数据的修改操作要么全部成功，要么全部失败。也就是说，事务的操纵序列要么完全应用到数据库，要么完全不影响数据库。这种特性称为原子性。
- 一致性：事务的一致性指的是，在事务执行之前和执行之后数据库都必须处于一致状态。一致性处理数据库中对所有语义约束的保护。假如数据库的状态满足所有的完整性约束，就说数据库是一致的。例如，当数据库处于一致性状态 S1 时，对数据库执行事务，在事务执行期间假定数据库的状态是不一致的，当事务执行结束时，数据库处在一致性状态 S2。
- 隔离性：隔离性指并发的事务是相互隔离的。即事务内部的操作及其正在操作的数据必须封锁起来，不被企图进行修改的事务看到。隔离性是 DBMS 针对并发事务间的冲突提供的安全保证。DBMS 可以通过加锁在并发执行的事务间提供不同

级别的隔离。如果对并发交叉执行的事务没有任何控制，那么操纵相同的共享对象的多个并发事务的执行可能会引起异常情况。这个概念主要在锁和并发中用到，在备份与恢复部分用不到这个概念，我们会在《高级进阶 DB2(第 2 版)》的“第 6 章：锁和并发”中讲解。

- 永久性：永久性意味着一旦事务提交，DBMS 就保证事务对数据库中数据的改变应该是永久性的，即使数据库发生存储介质故障。永久性确保已提交事务的更新不会丢失，即对已提交事务的更新能恢复。

在 DB2 数据库中，事务是最小的交易单位和恢复单位。事务由一条或多条 SQL 语句组成，最后是一条 COMMIT 或 ROLLBACK 语句。事务中的所有语句被看做单元，事务中包含的 SQL 语句要么全部成功，要么全部失败，以确保整个事务中数据的原子性和一致性。例如，客户试图将 1000 元从储蓄账户转到支票账户，在这种情况下，事务是这样的：

```
DELETE 1000 yuan from SAVINGS  account
INSERT 1000 yuan  to  CHECKING account
COMMIT
```

如果这些语句没有被当作单元，那么可以想象一下：在 DELETE 之后、INSERT 语句之前系统突然停电，会出现什么情况？这个客户将丢失 1000 元。但是，如果将这些语句当作单元，就不会发生这样的事情。DB2 将知道这个事务单元没有完成(COMMIT)，因此会回滚(rollback)之前语句做出的所有更改，并将受影响的数据行恢复到事务开始之前的状态。

COMMIT 或 ROLLBACK 用来显式地提交或回滚事务。就像上面我们举的那个例子，在这个事务中有两条 SQL 语句，这两条 SQL 语句要么全部成功，要么全部失败以保证交易的原子性。假设在执行完第一条 SQL 语句后突然机器停电了，那么数据库会自动执行 ROLLBACK，撤销(undo)前面已经执行的第一条 SQL 语句，这个工作由数据库自动来完成。如果在这两条 SQL 语句执行后，我们显式地执行了 COMMIT，那么对数据库的更改将永久生效，这就叫交易的永久性。假设我们提交后，更改的数据还没有来得及写到硬盘上，突然机器又停电了，这个没关系，因为一旦提交(COMMIT)，我们对数据库所做的操作就已经记录到数据库日志中了，数据库下一次重新启动时，会自动做 redo 操作。redo 就是从数据库日志中把刚才已经提交但是还没有来得及写到硬盘的数据更改重新再做一次。这个操作也是由数据库自动来完成的。当然，如果你的数据库日志也损坏了，那就没有办法执行 redo 操作了。

恢复的类型

了解完事务的概念后，接下来让我们看看针对上面所讲的几种故障场景，DB2 数据库中的几种恢复类型。

7.1.1　崩溃恢复(Crash Recovery)

对数据库执行的事务(也称工作单元)可能被意外中断。如果在发生故障时存在已开始但未提交的事务，或者存在已提交但未写到数据库中的事务，那么数据库就会处于不一致和不可用的状态。崩溃恢复的主要工作内容就是将数据库恢复为一致并可用状态的过程。为此，需要回滚未提交的事务，并重做当发生崩溃时仍在内存中的已提交事务，如图 7-1 所示。当数据库处于一致并可用状态时，它处于一种被称为“一致点”的状态。

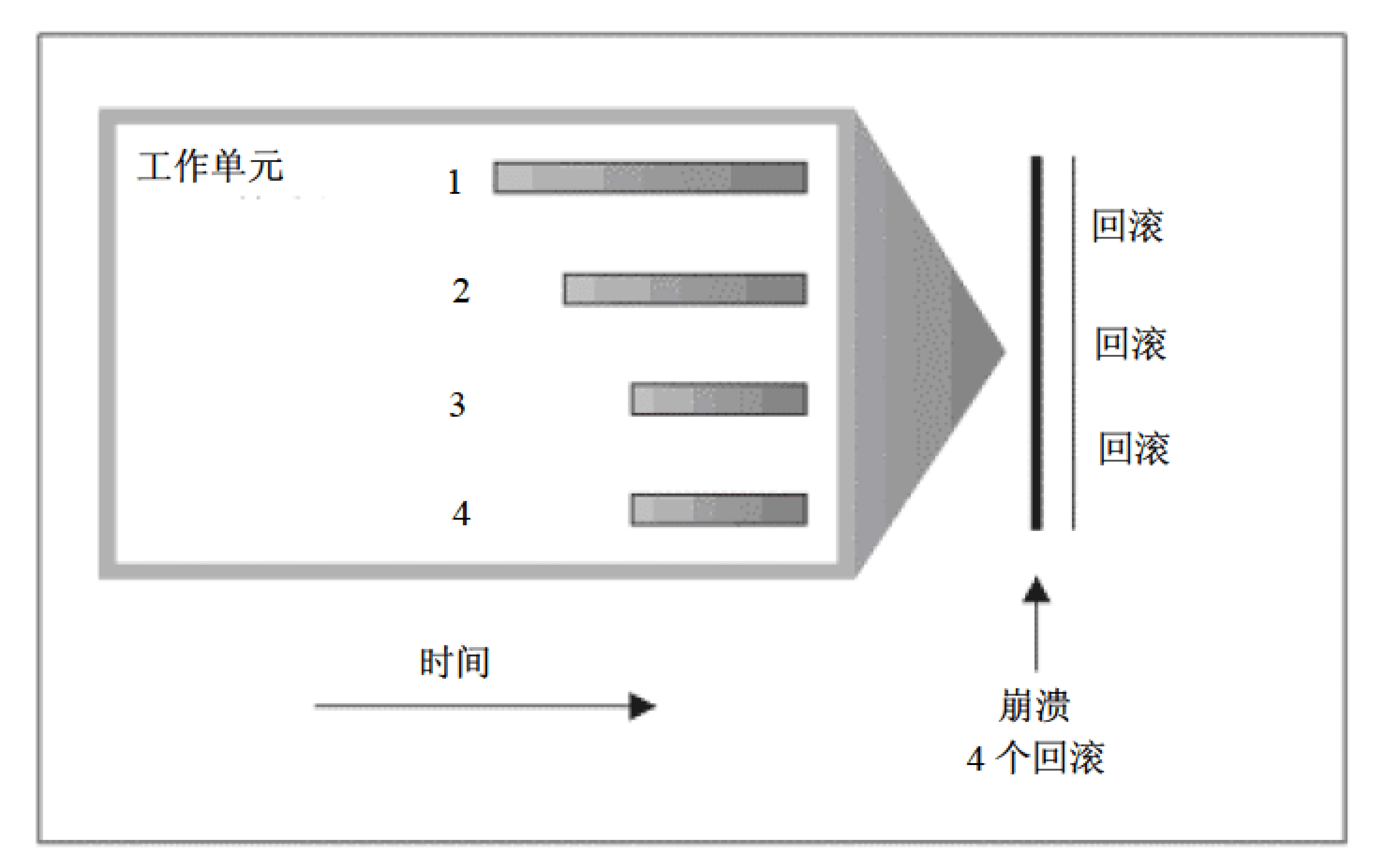


图 7-1　回滚工作单元(崩溃恢复)

可以想象，如果没有崩溃恢复这一功能，当数据库发生异常宕机时，后果将是整个数据库报废或丢失宕机发生时的数据，这将是多么可怕的情况。尤其是发生数据库异常宕机这种情形并非极为罕见。

如果希望崩溃恢复(不完整工作单元)的回滚是由数据库管理器自动完成的，那么应将数据库配置参数 *autorestart* 设置为 ON(这是默认值)以启用该自动重新启动参数。如果不想重新启动行为，可以将 *autorestart* 数据库配置参数设置为 OFF，但这样一来，就需要在执行崩溃恢复时主动发出 RESTART DATABASE 命令。

需要注意的是，数据库在宕机后由于处于不一致的状态，整个数据库都是不可用的，而恢复这一状态的唯一方式就是执行崩溃恢复。也就是说，崩溃恢复是恢复数据库为可用状态的必需环节。因此，强烈建议保留 autorestart 参数设置为默认值 ON。在设置为 ON 的情况下，我们第一次连接数据库，数据库就会自动启动崩溃恢复处理，直到崩溃恢复完成，数据库才可以使用。可以使用查看工具，观察崩溃恢复的处理过程，如 db2pd -util 或 db2 list utilities show detail 等。

通过 undo(回滚)未提交的事务，可以使处于不一致状态的数据库恢复到一致状态。再次考虑我们前面举的例子。如果在 COMMIT 语句之前出现停电事故，那么下一次 DB2 重新启动并访问数据库时，DB2 将首先回滚 INSERT 语句，然后回滚 DELETE 语句(回滚语句的顺序与这些语句当初执行的顺序相反)。

如果在崩溃恢复期间个别表空间发生错误，那么会让该表空间脱机，崩溃恢复继续进行。直到修复该表空间后才能对其进行访问。在崩溃恢复完成时，可以与该数据库建立连接，并且该数据库中的其他表空间将是可访问的。但是，如果脱机的表空间包含系统目录，那么必须先修复才允许连接数据库。

7.1.2 灾难恢复(Disaster Recovery)

术语“灾难恢复”用于描述在发生火灾、地震、恶意破坏或其他大灾害时复原数据库所需要执行的活动。灾难恢复通常指的是包含一系列应急恢复步骤的方案，而不是一项产品或单个技术、功能的使用。灾难恢复计划通常可以包括以下其中一项或多项：

- 保障数据同步的技术措施
- 在紧急情况下使用的场所
- 用于恢复数据库和应用的另一批机器
- 以非现场方式存储数据库备份和/或表空间备份以及归档日志
- 应急处理的流程及验证办法

通常说的灾备中心便是完成这一功能的场所。

在数据库级别保护数据可用性或防止单点故障的一种方法是实现 DB2 高可用性灾难恢复(HADR)功能。使用此功能后，HADR 通过将数据更改从源数据库(称为主数据库)复制到目标数据库(称为备用数据库)来防止数据丢失。也可以使用存储级别的映像功能(例如远程复制 PPRC 或 SRDF)来保护数据。PPRC 或 SRDF 提供了卷或磁盘同步复制功能来预防灾难。

注意：

灾难恢复对于重要系统来说至关重要。灾难恢复属于高可用的概念，这超出了本书的讲解范围。

7.1.3 版本恢复(Version Restore)

当数据库因发生故障导致无法使用时，我们可以采用“版本恢复”功能(或是为了重新搭建一个特定版本数据的数据库)。版本恢复指的是使用备份操作期间创建的映像来复原数据库的先前某个版本。版本恢复允许我们从 BACKUP 命令创建的备份映像将数据库恢复至前一个版本。被恢复的数据库将包含执行 BACKUP 命令时该数据库所处状态的信息。备

份之后执行的活动信息将丢失。对不可恢复数据库(即没有设置归档日志功能的数据库)只能使用这种方法。对于可恢复数据库(即设置了归档日志功能的数据库)还可对 RESTORE DATABASE 命令使用 WITHOUT ROLLING FORWARD 选项，这样在恢复完成后还可以继续使用前滚日志的方式进一步恢复数据。数据库备份使你可以将数据库复原到与执行备份时完全相同的数据和状态。但是，从备份时间到故障时间之间的所有事务都将丢失(请参阅图 7-2)。如果想要恢复此部分的数据库，就必须依靠前滚恢复技术来完成。

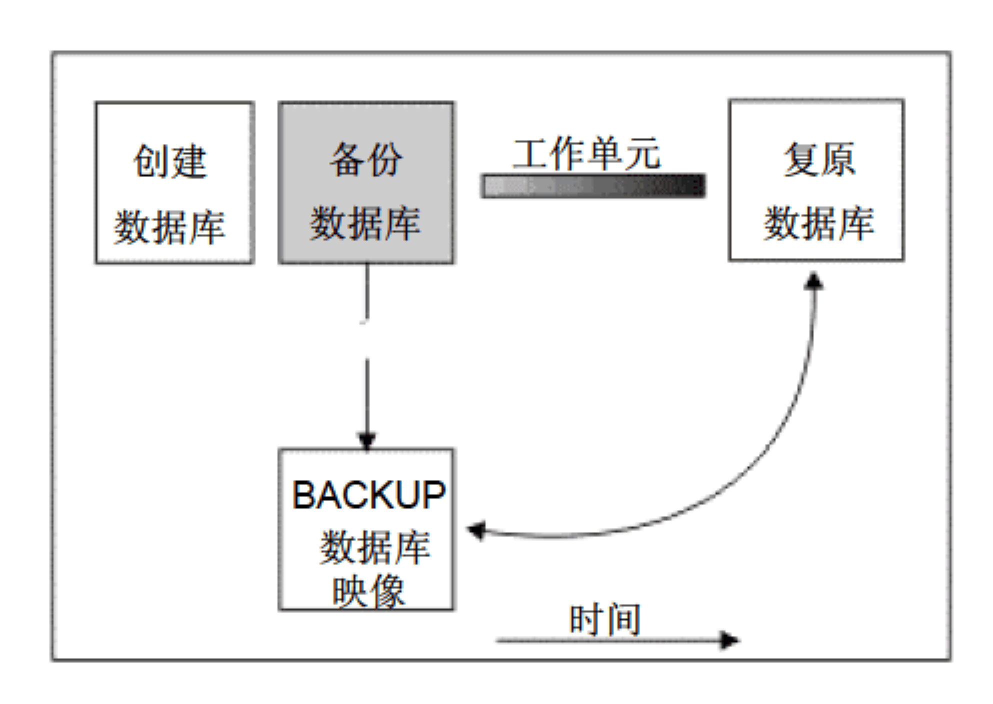


图 7-2　从进行备份开始到发生故障时为止的所有工作单元将丢失

使用版本恢复方法，为了保证有可用的数据库备份，就必须定期安排和执行完整数据库备份。

7.1.4　前滚恢复(RollForward Recovery)

如果发生存储故障或者用户不小心误删除了表中的重要数据，并且希望能恢复到最近的数据正确的时间点，那么在这种情况下需要用到前滚恢复。要使用前滚恢复方法，必须基于已经创建的数据库备份，并且已启用归档日志(方法是将 *logarchmeth1* 或 *logarchmeth2* 配置参数设置为 OFF 之外的值)。恢复数据库时，并且不使用 WITHOUT ROLLING FORWARD 选项，数据库在恢复操作结束时将处于前滚暂挂(Roll Forward Pending)状态。在这种状态下，可以对数据库采用前滚恢复。

有两种前滚恢复类型：

- 数据库前滚恢复。在此类型的前滚恢复中，DB2 会将记录在数据库日志中的所有事务应用到数据库中(请参阅图 7-3)，由于数据库日志记录了对数据库所做的所有更改，所以数据库中的数据将恢复到完整的一致状态。这种方法会将数据库恢复到在某特定时间点的状态，或者恢复到故障前的状态(即恢复到活动日志的末尾)。

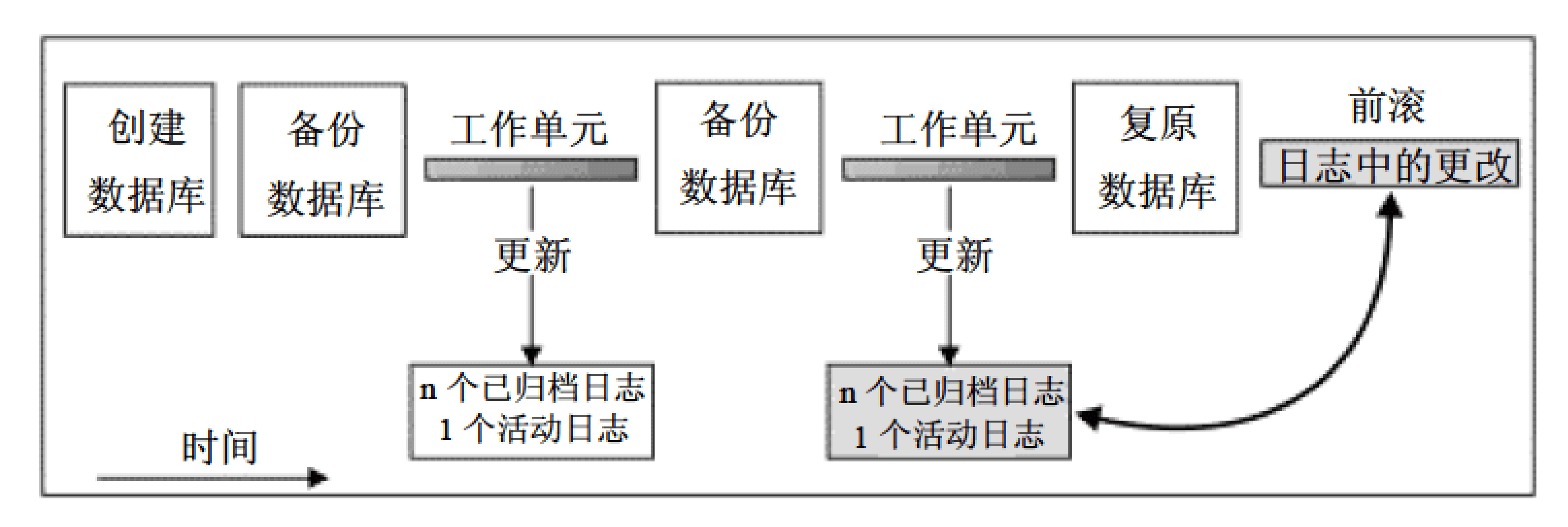


图 7-3　数据库前滚恢复(在运行时间较长的事务中，可以有多个活动日志)

● 表空间前滚恢复。在具备前滚条件的数据库中，既可以对整个数据库进行前滚恢复，也可以对某个表空间进行备份、恢复并前滚恢复(请参阅图 7-4)。要执行表空间恢复和前滚操作，需要整个数据库(即所有表空间)或个别表空间的备份映像。还需要前滚表空间过程中用到的数据库日志。

在所有的前滚操作中，一般可以选择在日志中前滚至以下两点之一：

◊ 日志末尾(to end of log)

◊ 某个特定时间点(称为时间点恢复)

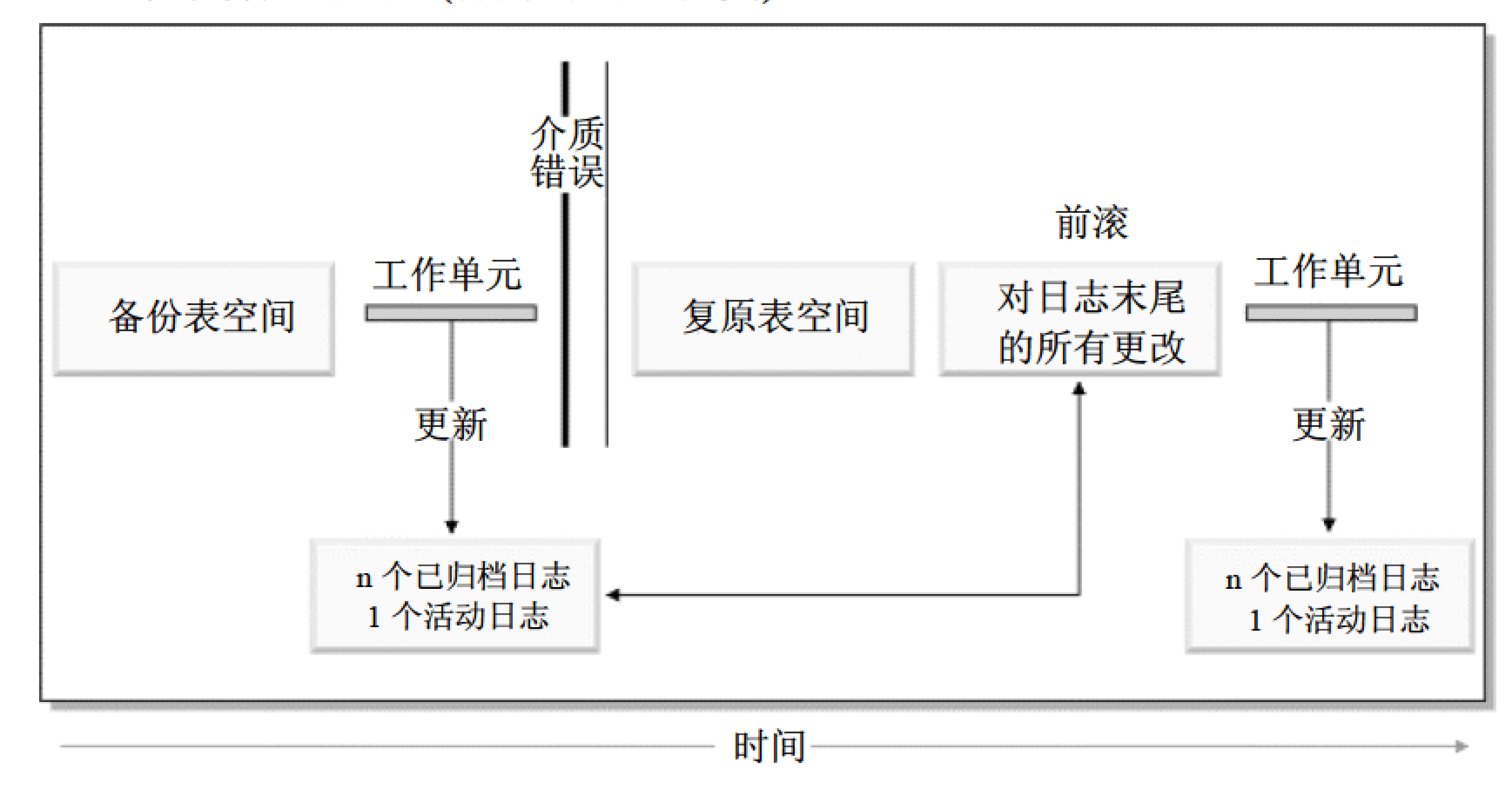


图 7-4　表空间前滚恢复(在运行时间较长的事务中，可以有多个活动日志)

可在下列两种情况下使用表空间前滚恢复：

● 在执行表空间恢复操作后，表空间始终处于前滚暂挂状态，并且必须前滚。调用 ROLLFORWARD DATABASE 命令将日志应用于表空间以使其前滚至某个时间点或日志末尾。

- 如果一个或多个表空间在崩溃恢复后处于前滚暂挂状态，那么首先应修复该表空间的问题。某些情况下，修复表空间的问题不涉及恢复数据库操作。例如，掉电或某些误操作可能导致表空间处于前滚暂挂状态。在这种情况下，恢复数据库操作并不是必需的。一旦修复表空间的问题，就可以使用 ROLLFORWARD DATABASE 命令将日志应用于表空间，使其恢复到日志末尾。如果在进行崩溃恢复之前解决了表空间的故障原因，那么崩溃恢复操作足以使数据库恢复到一致并且可用的状态。

注意：

如果出错的表空间包含系统编目表(Catalog Table)，将不能启动数据库。必须恢复 SYSCATSPACE 表空间，然后执行前滚恢复直至日志末尾。

崩溃恢复结合使用完整的数据库备份和日志文件，将数据库在版本恢复的基础上再执行数据恢复，从而扩展了版本恢复。因此使用崩溃恢复的必备条件是，必须首先创建数据库的完整备份，然后在备份上才能再应用日志以完成崩溃恢复。这个过程允许将数据库或表空间恢复到某个特定的时间点上。另外，前滚恢复还要求数据库启用归档日志功能，按默认选项创建的数据库并未启动这项设置，这就需要我们手工设置，而且我们也强烈建议在生产系统中启用这一设置。

不同的恢复类型对日志的要求不同，下面将介绍数据库日志。

7.2　DB2 日志

数据库备份是数据库的完整副本。其实可以这样简单理解——backup=copy，备份其实就是给数据库做某个特定时刻的 copy。那么恢复呢？恢复本质也是 copy 操作，恢复的本质就是恢复到我们备份时刻的数据。那么假设我们周一晚上做了数据库的备份，周二中午 12 点数据库存储介质出现了故障。如何能够实现恢复呢？那么我们首先是把数据库恢复到我们备份的那个时刻(也就是周一晚上)，但是备份之后和周二 12 点之间对数据库已经做的事务怎么办呢？这就需要用到数据库日志，因为一旦事务提交，我们对数据库做的修改操作(insert、update 和 delete 等)都会记录到数据库日志中。所以我们就可以用数据库日志(前提是数据库日志没有受到损坏)把备份之后和数据库崩溃之前的所有修改操作重做(redo)一遍。这就是数据库前滚恢复的原理。所以大家可以看到在这个过程中，数据库的日志至关重要。下面我们来详细介绍 DB2 日志。

7.2.1 日志文件的使用

为了确保用户数据的完整性，DB2 使用了提前写日志存档模式。提前写日志存档的基础是指：当发出删除、插入或更新数据库中某一数据的 SQL 调用时，所做的数据变更首先要写到日志文件中，当发出一条 SQL COMMIT 命令时，DB2 需要把数据写入表中，而在 DB2 接收到 COMMIT 命令之后且在数据开始写入到表中之前，DB2 要保证已经把为了重做(redo)所需要的日志文件都写入磁盘中(还有其他的事件会触发日志被写入磁盘，后续我们会进一步讨论)。这就保证了日志可以用于恢复任何已经 COMMIT 的事务。在发生断电之类的不幸事故时，日志文件可以用来把数据库退回到原来的某个一致性状态。所有被提交确认的事务都将重新再做一遍，所有未提交确认的事务都将退回到原有起点。图 7-5 展示了这种模式。

在图 7-5 中，一共执行了 4 条 SQL 语句。这些语句被缓存在程序包缓存中，数据页被从硬盘取出到缓冲池中。随着 SQL 语句的执行，更改首先被记录到日志缓冲区中，然后被写到日志文件中。在这个例子中，更改过的已经提交的数据页(脏页)还没有被写到硬盘上。

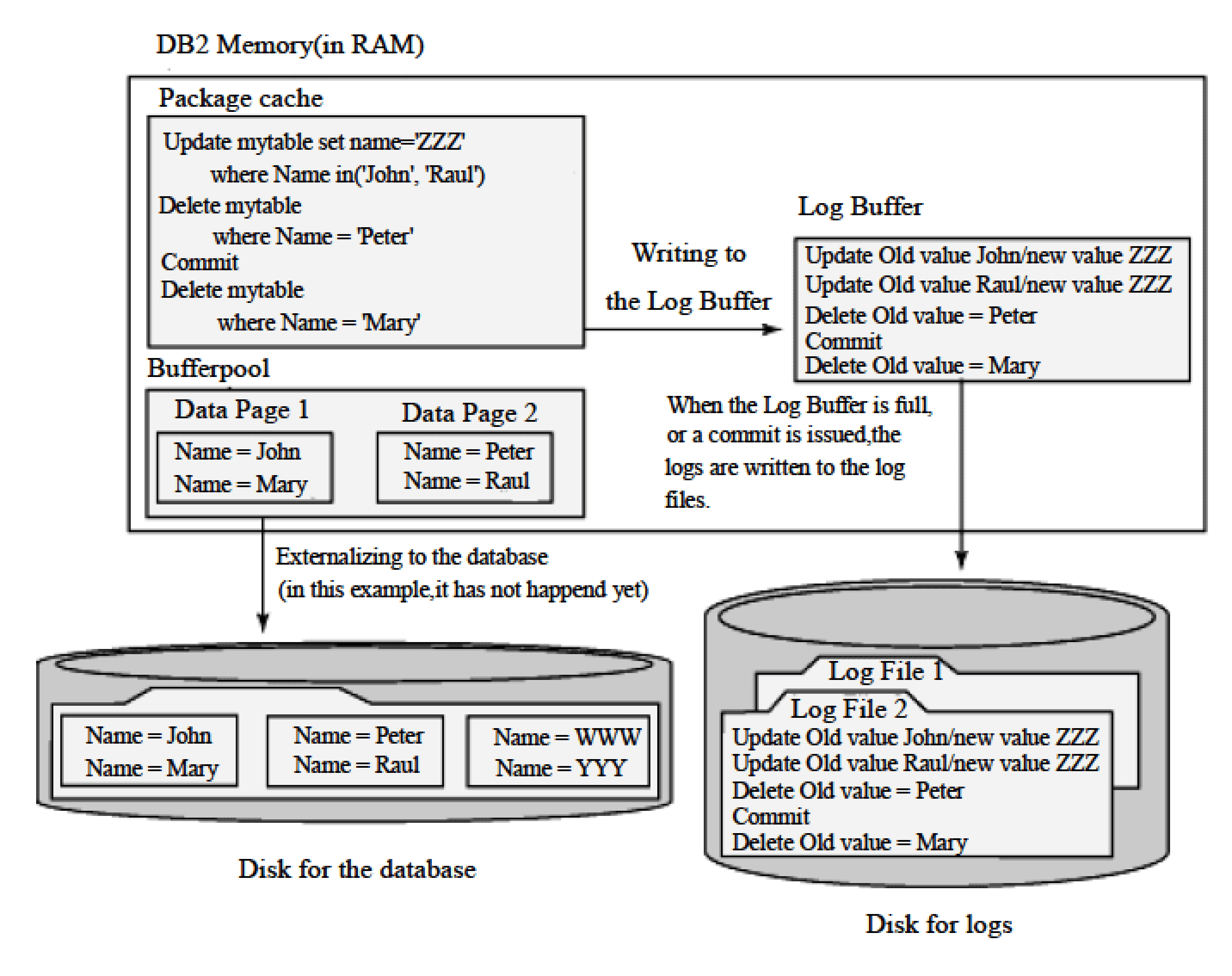


图 7-5　日志文件的使用

日志中只记录 DML 操作(insert、update 和 delete 操作)，假设我们在前台发出了一条 insert、update 或 delete 语句，那么在日志中就相应地记录这条 SQL 语句的 redo 和 undo 操作，undo 是保证能够执行回滚(rollback)操作；而 redo 是保证数据库可执行前滚(roll forward)

操作。下面我们来看看 DB2 日志中主要记录哪些内容，请参阅表 7-1。

表 7-1　DB2 日志的内容

时间戳	LSN	用户名	运算	所有者	表	SQL 重做(redo)	SQL 还原(undo)
26-十月-2012 10:03:33	24814	PROD	DDL	PROD	T1	create table prod.t1(id int)	注意：DDL 操作，比如 create、drop 和 alter 操作，不记录日志
26-十月-2012 10:03:33	24818	PROD	COMMIT			Commit	
26-十月-2012 10:03:45	24822	PROD	INSERT	PROD	T1	insert into "PROD"."T1" values "ID" = 1	delete from "PROD"."T1" where "ID" = 1 and RID = 'AAAGMNAAEAAABrHAAA'
26-十月-2012 10:04:00	24828	PROD	DELETE	PROD	T1	delete from "PROD"."T1" where "ID" = 1 and RID = 'AAAGMNAAEAAABrHAAA'	insert into "PROD"."T1" values "ID" = 1
26-十月-2012 10:04:22	24836	PROD	UPDATE	PROD	T1	update "PROD"."T1" set "ID" = 200 where "ID" = 2 and RID = 'AAAGMNAAEAAABrHAAB'	update "PROD"."T1" set "ID" = 2 where "ID" = 200 and RID = 'AAAGMNAAEAAABrHAAB'
26-十月-2012 10:04:23	24838	PROD	COMMIT			commit	

通过上面的日志我们可以看到，当在前台发出 insert、delete 或 update 语句时，日志中同时记录 redo 和 undo 操作以保证数据库的前滚和回滚。大家注意，前滚(roll forward)这个词和 redo 其实是等同的，回滚(rollback)和 undo 是等同的。

我们还需要了解 LSN 的意义，LSN 是 Log Sequence Number 的缩写，直译为日志顺序号。LSN 是日志文件中日志记录的唯一标识，在数据库中 LSN 永远不会重复，这样就可以通过 LSN 来精确定位日志记录。在数据库中出现日志相关的故障或者在复杂场景下使用日志恢复数据库时，LSN 显得特别重要，可以用于判断很多日志中的问题。当然，使用这一信息来定位问题通常只能由资深的 IBM 支持人员才能完成。可以使用 get snapshot 或 db2pd -logs 命令来观察这些信息。

所有数据库都有与它们相关联的日志文件。日志文件有预先定义的大小(LOGFILSZ)。因此，当日志文件被填满时，日志存档过程就要在另一个日志文件中继续进行。

7.2.2 日志类型

数据库中有两类日志：

- 循环日志(circular logging)
- 归档日志(archival logging)

1. 循环日志

循环日志是 DB2 默认的日志记录模式。顾名思义，这种类型的日志记录以循环模式重用日志。例如，如果有 4 个主日志，DB2 将按照以下顺序使用它们：Log #1，Log #2，Log #3，Log #4，Log #1，Log #2，依此类推。

在循环日志记录模式下，只要这个日志文件中包含的数据已经全部提交且已写到磁盘上，这个日志文件就可以被重用。换句话说，如果日志仍然是活动日志，就不能被重用。循环日志使用两种日志文件：

- 主日志文件(primary log files)
- 辅助日志文件(second log files)

主日志文件是预先分配的，也就是在数据库启动过程中创建，是数据库启动的一部分(如果在启动数据库之前曾经创建过这些日志，就不会再次创建这些日志，而是直接使用已经存在的日志，比如重启数据库)，而辅助日志文件仅在需要时才分配(但分配后将不会被主动回收，除非重启数据库)。如果数据库需要下一个日志文件，并且主日志文件已经用完且不能被重用，那么将分配辅助日志文件，直至主日志文件变得可供重用或者分配的辅助日志文件的数目达到限制为止。

主日志文件和辅助日志文件的数目由数据库参数 LOGPRIMARY 和 LOGSECOND 来决定。

仍然使用前面循环日志记录的例子，如果有运行时间很长的事务，这个事务要横跨 5 个(图 7-6 中默认配置了 4 个主日志文件)日志文件，那么会出现什么情况呢？在这种情况下，DB2 会再多分配一个日志文件——一个辅助日志文件。图 7-6 展示了其中的原理。

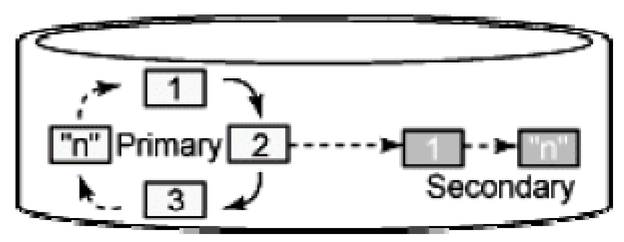


图 7-6 循环日志的原理

当数据库最初被创建时，循环日志方式作为默认的日志方式被激活。

循环日志记录数据库仅能恢复到曾经做过备份的点。对数据库进行恢复时，备份点之后对数据库做的所有工作都将丢失。由于这个原因，循环日志方式最适合用于那些只供查

询的数据库或开发测试类型的数据库，重要的生产数据库不建议使用。

2. 归档日志

当使用归档日志记录模式时，数据库会经常归档(保留)日志。在循环日志记录模式下，已提交且被写到磁盘上的日志文件将被重用；而在归档日志记录模式下，这些日志文件将得到保留。例如，如果有 4 个主日志文件，DB2 将按照以下顺序使用它们：Log #1，Log #2，Log #3，Log #4(如果 Log #1 的所有事务已被提交且已写到磁盘上，将归档 Log #1)，Log #5(如果 Log #2 的所有事务已被提交且已写到磁盘上，将归档 Log #2)，Log #6，依此类推。也就是在使用完一个日志文件后，会创建并使用新的日志文件，并同时将最早的已经使用完成的日志文件归档。

正如这个例子演示的那样，DB2 将保持 4 个主日志文件可用，即使一些日志文件中填满了已被提交且已写到磁盘上的数据对应的日志，DB2 也不会重用它们。DB2 不会覆盖已经成为归档日志的日志。图 7-7 阐释了其中的原理。

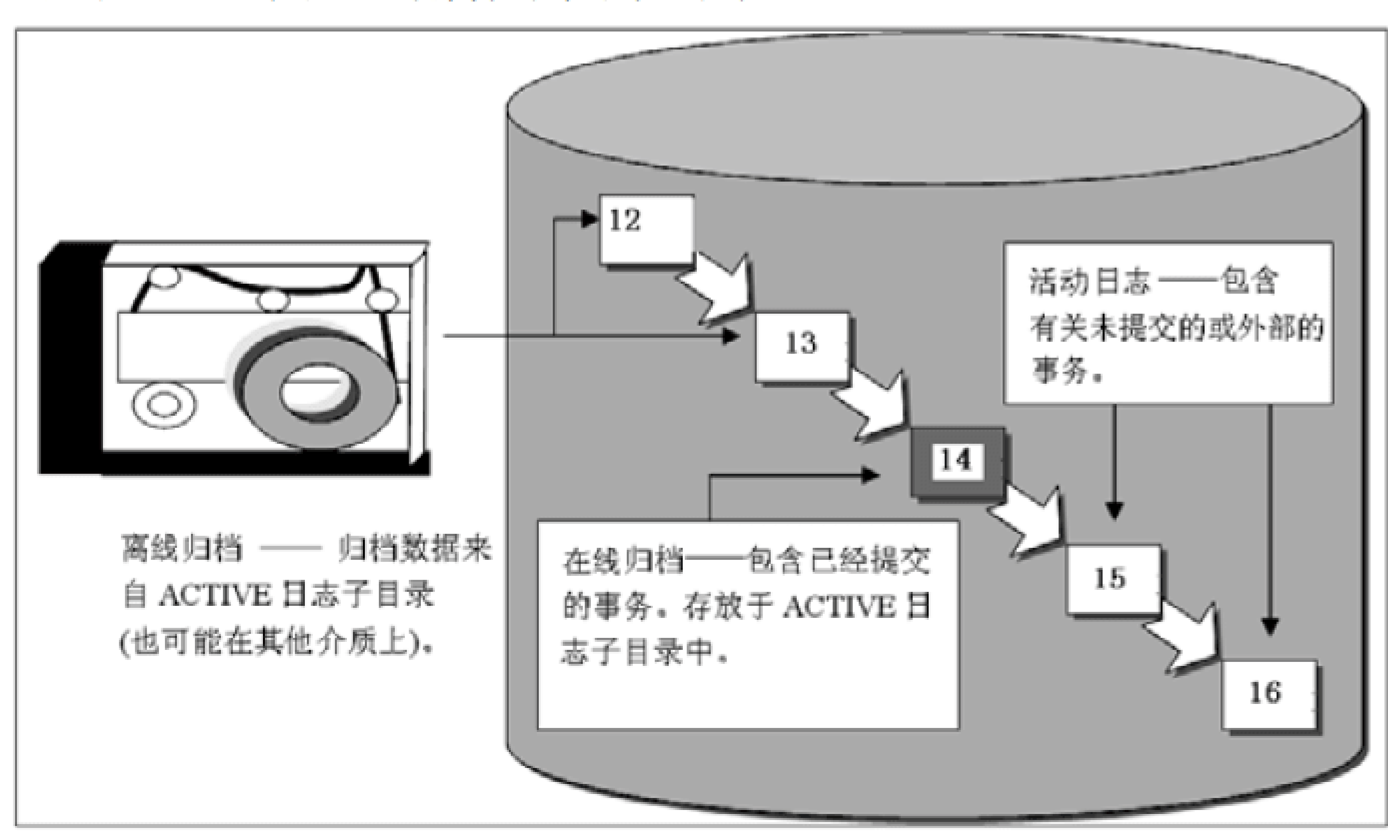


图 7-7　归档日志的原理

活动日志文件(由数 15 和 16 表示)

如果某个日志文件满足以下两个条件中的任意一个，就属于活动(active)日志文件：

- 包含尚未提交或回滚的事务信息
- 包含已经提交但是更改尚未写到磁盘上的事务信息

注意：

活动日志特别重要，如果活动日志丢失，那么数据库将无法启动。

在线归档日志(由数 14 表示)

包含已提交且已写到磁盘上的日志。这些日志与活动日志放在相同的目录中。

离线归档日志(由数 12 和 13 表示)

已经从活动日志目录转移到另一个目录或媒介上的归档日志。这种移动可以手动地完成，也可以由 DB2 自动完成。

数据库中日志记录的类型是由数据库参数 LOGARCHMETH1 决定的。当 LOGARCHMETH1 为 OFF(默认值)时，归档日志记录被禁用，循环日志记录被启用。

为了启用归档日志记录，可以将 LOGARCHMETH1 设置为以下值中的任何一个：

- LOGRETAIN　日志文件将被保留在活动日志目录中，一般这表示已归档日志文件的工作需要 DBA 手工完成。
- USEREXIT　日志的归档和检索是由用户提供的用户出口程序自动执行的，这个出口程序必须由 db2uext2 调用。这个程序用于将在线归档日志移动到与活动日志目录不同的目录中，或者移动到另一媒介上。当在 ROLLFORWARD 操作期间需要某些离线归档日志时，这个程序还可以用于将离线归档日志取出到活动日志目录中。在 Windows 下，db2uext2 必须存放在 sqllib\bin 目录中；在 UNIX 下，db2uext2 必须存放在 sqllib/adm 目录中。这是 DB2 V8 之前版本的主要归档方法。
- DISK:<directory_name>　通常，<directory_name>是绝对路径，此设置使用与 USEREXIT 类似的方法。此时，DB2 不调用用户出口程序，而是自动将归档日志文件从活动日志目录移动到由参数指定的目录中。此设置简单易用，目前已成为主流的日志归档方法。
- TSM:[management class name]　使用与 USEREXIT 相同的算法。日志被归档到本地 Tivoli Storage Manger(TSM)服务器上。management class name 参数是可选的，如果没有指定该参数，将使用默认的管理类。
- VENDOR:<library_name>　使用与 USEREXIT 相同的算法。日志使用指定供应商的库来归档，参数<library_name>的实际值一般是由第三方提供的库文件的绝对路径。如果归档到由备份管理软件 NBU(Network BackUp)管理的磁带库上，就可以使用这种方法。

由于向后兼容的原因，数据库配置文件仍然包含参数 LOGRETAIN 和 USEREXIT。从 DB2 V8.2 开始，这两个参数已经被 LOGARCHMETH1 取代。如果更新 USEREXIT 或 LOGRETAIN 参数，那么 LOGARCHMETH1 将自动被更新，反之亦然。

归档日志方式不是默认的日志工作方式，但却是允许用户执行前滚(roll forward)恢复的唯一方法，而且很多其他重要的数据库维护功能需要在归档日志模式下才能完成，因此强烈建议在生产数据库上使用归档日志模式。

3. 无限日志记录

不管是使用循环日志记录还是归档日志记录，日志空间都可能被填满活动日志。如果启用无限日志记录，DB2 就会在日志被填满时立即归档这个日志，而不会等到日志中所有的事务都已经被提交且写到磁盘的时候才来归档日志。这样可以保证活动日志目录永远不会被填满。例如，如果有长时间运行的事务，在启用无限日志记录模式的情况下，就不会出现日志空间被耗尽的情况。

然而，我们不建议使用无限日志记录，因为可能延迟紧急事故的恢复时间，这是因为需要从归档站点检索活动日志。无限日志记录是归档日志记录的派生物。要启用无限日志记录，可以：

- 将 LOGSECOND 数据库配置参数设置为－1
- 启用归档日志记录

7.2.3　日志相关配置参数

DB2 的日志管理涉及一系列参数。很多参数在被修改后，需要重启数据库才能生效。表 7-2 列出了影响日志工作的各个参数。

表 7-2　与日志记录相关的一些数据库配置参数

参　　数	用　　途
LOGPRIMARY	表明要分配的主日志文件的数量
LOGSECOND	表明可以分配的辅助日志文件的最多数量
LOGFILSIZ	用于指定日志文件的大小(4KB 页的个数)
LOGBUFSZ	日志缓冲区大小参数决定分配多少内存空间作为缓冲区，在把日志记录写到磁盘之前，暂时将它们保留在日志缓冲区。这一参数的值的单位是 4KB 页
NEWLOGPATH	日志文件默认的子目录定义在数据库目录的子目录 SQLLOGDIR 中。出于恢复目的，最好把日志文件存放到与数据库文件不同的物理磁盘中。这一参数标识日志文件存放的新路径
SOFTMAX	软检查点，是个百分数，例如 50 表示日志文件写满 50%时，数据库执行 checkpoint 操作，把内存中的已经提交的数据(脏页)写到磁盘上
MINCOMMIT	组提交数，默认是 1，表示每次提交都写日志。假如设置为 5，就表示累计 5 次提交才写一次日志文件；如果没有累计到 5 次，那么每隔 1 秒写一次日志文件(只有在 MINCOMMIT 的设置大于 1 的情况下才有此机制)
LOGARCHMETH1	第 1 个日志归档方法
LOGARCHMETH1	第 2 个日志归档方法

(续表)

参　数	用　途
MIRRORLOGPATH	映像日志路径
TRACKMOD	启用增量备份

可以通过 update db cfg 命令来更改这些参数。例如，如果希望启用归档日志，可以执行：

```
db2 update db cfg for sample using LOGARCHMETH1 DISK:/logdir
```

在第一次启用归档日志时，数据库会处于 backup pending 状态，此时需要给数据库做全备份。

如果要启用增量备份，可以执行：

```
db2 update db cfg for sample using LOGARCHMETH1 DISK:/logdir trackmod on
```

更新归档日志文件的目标文件夹(为归档日志文件指定路径可以将归档日志模式打开)。

7.2.4　数据库日志总结

日志记录类型与恢复类型

现在你理解了不同类型的日志记录和恢复类型，但要注意的一点是，并不是所有日志记录类型都支持所有的恢复类型。循环日志记录只支持崩溃恢复和版本恢复，而归档日志记录则支持所有类型的恢复：崩溃恢复、版本恢复和前滚恢复。

可恢复数据库和不可恢复数据库

可恢复(*recoverable*)数据库是指可以使用崩溃恢复、版本恢复或前滚恢复进行恢复的数据库；因此，对于这些数据库，需要启用归档日志记录。不可恢复(*nonrecoverable*)数据库是指不支持前滚恢复的那些数据库；因此，只需要使用循环日志记录。

数据库日志总结

到目前为止，我们讲解了关于数据库日志和日志记录的一些概念。图 7-8 对这些概念作了总结。

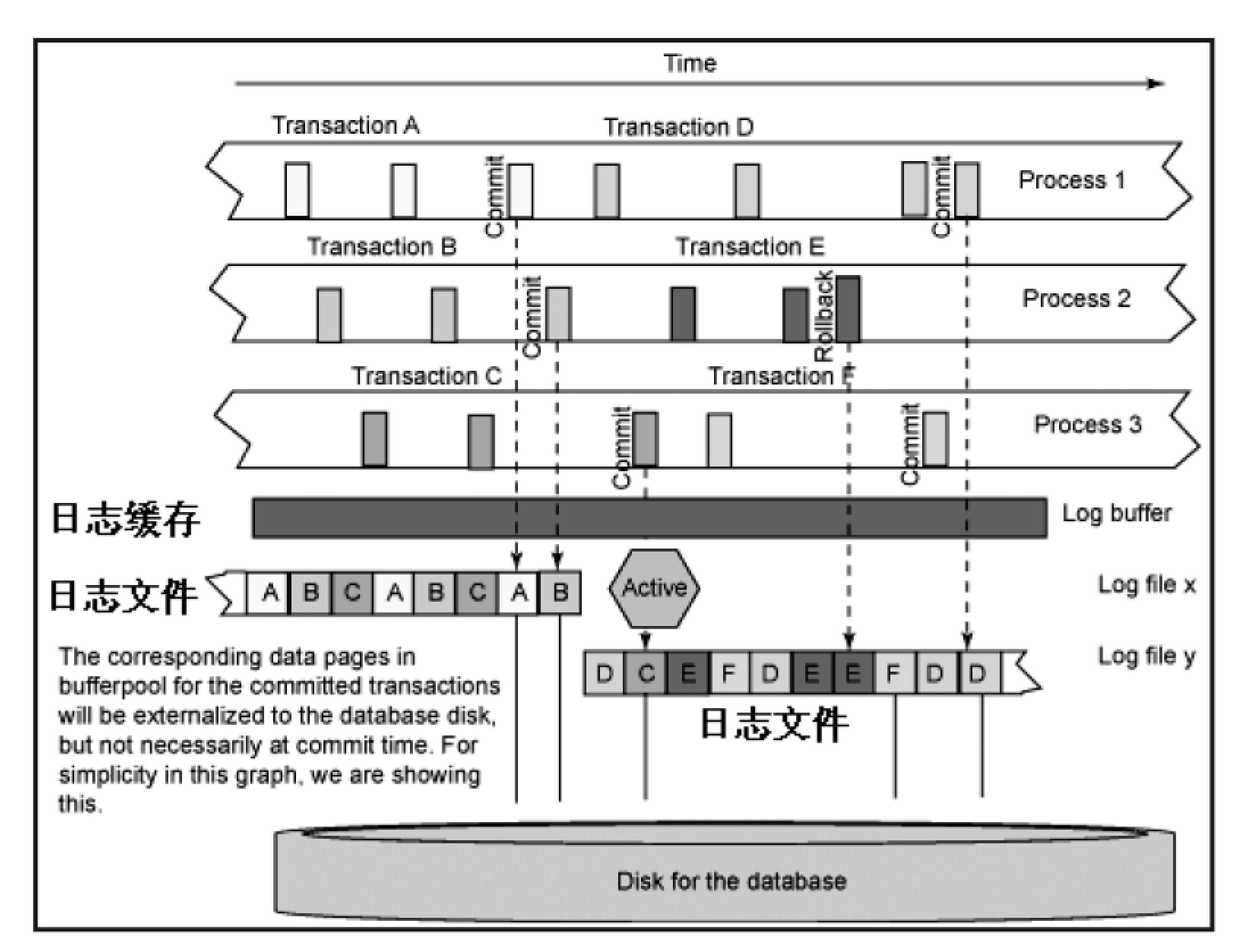


图 7-8　事务日志文件的使用

图 7-8 说明了如何使用多个日志文件来管理若干并行事务。图的顶部表示访问同一数据库的 3 个用户进程(1~3)在时间上的进展。方框表示数据库的变更，比如插入或更新。用户可以看到每个事务(A~F)的生存期。图的中下方表示数据库的变更如何被同步地记录到日志文件(x 和 y)中。每个小方格中的字母表示数据库变更属于哪个事务。

当发出一条 SQL COMMIT 命令或日志缓冲区被填满时，包含该事务的日志缓冲区便被写入到磁盘上。事务 E 在任何时候都不被写入磁盘，因为它使用 SQL ROLLBACK 命令来终止。当日志文件 x 因存放事务 B 的第一个数据库变更而用完了空间时，日志存档进程将转向日志文件 y 继续进行。日志文件 x 仍然保持活动状态，直到把事务 C 的所有变更都写入磁盘中为止。在将日志存档工作转向日志文件 y 的瞬间，日志文件 x 仍然保持处于活动状态的时间周期，用小的六边形来表示。

标有 *active* 字样的六边形表示日志文件 x 仍然被当作活动日志的时间。可以看到，这个六边形在表示日志文件 y 中事务 D 和事务 C 一部分的正方形之上。为什么日志文件 x 在被填满之后仍然被当作活动日志呢？这是因为它包含了还没有被提交和写到磁盘上的日志信息。日志文件 x 包含事务 A、B 和 C。只有事务 A 和 B 被提交(在这个例子中，它们也随之立即被写到磁盘上)；而事务 C 仍然在运行，并且被写到日志文件 y 中。当事务 C 在日志文件 y 中被提交时(在这个例子中，它也随之立即被写到磁盘上)，日志文件 x 将不再

被当作活动日志，它将成为在线归档日志。

大家最好记住日志中记录的是我们对数据库做的操作(insert、update 和 delete 等)，而表空间对应的容器上记录的是我们真正的数据。有人经常问我，日志和数据之间是什么样的增长关系。我在这里给大家举一个简单的例子：假设有一个表，表中有一个 INT 字段 seqno，我把这个字段 update 了 1000 次，这个表的数据并没有真正发生变化。但是数据库日志中会记录 1000 次 update 操作。所以日志和表数据量大小没有必然的关系，日志只和你做的 SQL 操作的多少有关系。

7.2.5 DB2 日志的建议设置

尽量使用归档日志模式

在前面的章节中，已经两次强烈建议在重要数据库上使用归档日志模式，因为只有这个设置才能与数据库备份配合，达到将数据库恢复到任何时间点的目标。

日志大小，主、辅日志数量的设置考虑

数据库的 3 个参数——LOGFILSIZ、LOGPRIMARY 和 LOGSECOND 共同决定了数据库中日志的可用空间的范围。

每个日志的大小为 LOGFILSIZ×4KB。其中数据库启动时就分配的日志大小为 LOGFILSZ×LOGPRIMARY×4KB，数据库可用的最大日志空间为 LOGFILSIZ×(LOGPRIMARY+LOGSECOND)×4KB。这 3 个参数的默认设置较小，分别为 1000、3、2。按上面公式进行计算：1000×(3+2)×4KB=5000KB。这个关于数据库可用日志空间的默认设置对于一般系统来说是偏小的，因此在创建数据库后首先需要修改这 3 个参数。注意 LOGFILSZ 的限制为 4GB，LOGPRIMAY+ LOGSECOND 的设置不能超过 256。

数据库日志相关的实际结果可通过数据库快照获得，比如：

```
db2 get snapshot for database on <dbname>
```

有关的输入如下：

```
……..
Log space available to the database (Bytes)= 130743006165
Log space used by the database (Bytes)     = 210721835
Maximum secondary log space used (Bytes)    = 0
Maximum total log space used (Bytes)       = 4074688903
Secondary logs allocated currently         = 0
……..
```

日志文件大小(LOGFILSIZ)的设置范围可以非常宽泛，从几 MB 到几 GB 都可能是适当的设置，只要综合考虑可用日志空间的大小满足需求即可。有一点需要注意的是，日志文件越大，初始化新的日志文件所需要的时间就越长，这可能会引起性能问题。

一般情况下，考虑设置主日志的数量需要能够满足日常使用需求，并预留一定的富余空间即可，设置辅助日志数量的一般考虑因素是在特殊情况下能够提供足够的缓冲，从而在负载出现波动的情况下，不至于将数据库日志空间用尽。

但在对响应时间有非常高要求的交易型数据库中，建议将主日志的数量增大。因为当主日志的空间用尽而需要创建新的日志文件时，数据库会出现数据写被短暂挂起，直至新的日志文件创建完成，数据库才能继续写入数据。

日志缓冲的考虑

用于指定存放日志缓冲大小的参数是 LOGBUFSZ，其单位也是 4KB。日志缓冲的主要目的是以尽量多的异步 I/O 代替同步 I/O，从而提高日志写的效率。此参数的默认值在 DB2 V9.5 中是 8，在 DB2 V9.7 和 DB2 V10.1 中是 256。DB2 V9.5 中的默认设置在大多数情形下是偏小的，因此需要调整此参数的设置为较高的值，比如 128 或 256。对于 DB2 V9.7 和 V10.1 中的默认设置 256 来说，一般已经足够，除非有特殊需求，否则无须调整。

提高写日志的性能

对于 DB2 中的事务来说，在与事务对应的日志成功写入日志文件后，事务才能完成 COMMIT 操作。因此，如果日志写入的性能慢，势必会影响事务的性能。因此在压力比较大的系统中，应该考虑优化日志写入的性能。

评估日志写性能的指标可以按以下方法获得：

```
$db2 get snapshot for db on testdb|more

              Database Snapshot
.........
Log write time (sec.ns)                    = 25512.063380216
Number write log IOs                      = 26347156
.........
```

日志写的平均时间= Log write time/Number write log IOs=25512.063380216/26347156=0.968 ms

一般情况下，小于 5ms 的值应该属于正常范围内。

优化日志写的性能可以考虑下面几个因素：

- 适当设置 LOGBUFSZ，此参数的设置原则可以参考上面的章节。

- 日志文件所在的文件系统与数据库中数据使用的存储尽量分开或分散。
- 日志文件所在的文件系统如果使用多个LUN或磁盘，那么应尽量使用条带技术(Stripe)来提高性能。此方法对日志写性能的提高有非常大的帮助。

7.3 数据库和表空间备份

在线与离线

如果正在执行在线操作(备份、恢复、复原)，那么其他用户也可以同时访问我们正在操纵的数据库对象；如果正在执行离线操作，那么不允许任何其他用户同时访问我们正在操纵的数据库对象。在这一节中，我们会经常使用术语“在线”和“离线”。

7.3.1 数据库备份

数据库备份是数据库的完整副本。其实可以这样简单地理解——backup=copy，备份其实就是给数据库做某个特定时刻的 copy。除了数据外，备份副本还包含关于表空间、容器、数据库配置、日志控制文件和恢复历史文件的信息。注意，备份不会存储数据库管理器配置文件或注册变量。只有数据库配置文件才会得到备份。

为了执行备份，需要 SYSADM、SYSCTRL 或 SYSMAINT 权限。

下面是用于备份的 BACKUP 实用程序的语法：

```
    BACKUP DATABASE database-alias [USER username [USING password]]
    [TABLESPACE (tblspace-name [ {,tblspace-name} ... ])] [ONLINE]
    [INCREMENTAL [DELTA]] [USE {TSM | XBSA} [OPEN num-sess SESSIONS]
    [OPTIONS {options-string | options-filename}] | TO dir/dev
    [ {,dir/dev} ... ] | LOAD lib-name [OPEN num-sess SESSIONS]
    [OPTIONS {options-string | options-filename}]]
    [WITH num-buff BUFFERS] [BUFFER buffer-size] [PARALLELISM n]
    [COMPRESS [COMPRLIB lib-name [EXCLUDE]] [COMPROPTS options-string]]
    [UTIL_IMPACT_PRIORITY [priority]] [{INCLUDE | EXCLUDE} LOGS] [WITHOUT
PROMPTING]
```

让我们来看一些例子，进而了解其中一些选项的作用。

要为数据库 sample 执行完整离线备份，并将备份副本存储在 d:\backup 目录中，可以使用以下命令：

```
BACKUP DATABASE sample TO /backup
```

要使用其他一些常见选项为数据库 sample 执行完整离线备份，可以使用以下命令：

```
(1)   BACKUP DATABASE sample
(2)   TO /db2backup/dir1, /db2backup/dir2
(3)   WITH 4 BUFFERS
(4)   BUFFER 4096
(5)   PARALLELISM 4
```

接下来更仔细地观察前面的命令：

(1) 表明要备份的数据库的名称(或别名)。

(2) 指定用于存储备份的位置，可以指定多个路径，每个路径下的备份文件都会有序列号，例如 001、002 等。指定多个路径来保存备份映像可能有很多原因，尤其是这种方法可以通过并行 I/O 提高备份性能。

(3) 表明在备份操作期间可以使用多少个内存缓冲区。使用多个缓冲区可以提高性能。

(4) 表明每个备份缓冲区的大小。

(5) 决定使用多少并行读/写进程/线程来进行备份。

对于 DB2 V9 及以后的版本，建议大多数选项使用默认值，因为 BACKUP 实用程序会自动选择设置选项以获得最佳性能。

语法中没有关键字 OFFLINE，因为这是默认模式。如果要为 sample 数据库执行完整在线备份，就必须指定关键字 ONLINE。在线备份要求启用数据库的归档日志模式。下面的语句展示了如何执行在线备份：

```
BACKUP DATABASE sample  ONLINE  TO /backup
```

如果数据库的日志模式不是所需要的归档日志模式，在执行上面的命令时就会收到下面的提示信息：

```
SQL2413N  Online backup is not allowed because the database is not recoverable
or a backup pending condition is in effect.
```

由于在线备份允许用户在执行备份的同时访问数据库，因此这些用户作出的更改很可能不会包含在备份副本中。因此，仅仅凭借在线备份往往还不足以进行恢复到备份结束时间点的所有数据，另外还需要备份操作期间产生的日志文件。当在线备份完成时，DB2 强制关闭当前活动日志(并将它们归档)，因此很容易在备份完成时收集当前的日志。

为了将相关的日志也同时备份到数据库的备份映像中，可以使用 BACKUP DATABASE 命令的 INCLUDE LOGS 选项。这样可以确保即使丢失了日志，也仍然可以使用备份映像中包含的日志来恢复到一致点上。INCLUDE LOGS 选项只适用于在线备份。

例如，要对 sample 数据库和日志进行在线备份，并以/backup 作为目标目录，可以发出：

```
BACKUP DATABASE sample ONLINE TO /backup INCLUDE LOGS
```

7.3.2　表空间备份

如果数据库中只有一部分表空间有较多的数据变化，那么可以选择不备份整个数据库，而是只备份特定的表空间。这样就可以使用这个备份来恢复特定的表空间。

要执行表空间备份，可以使用以下语法：

```
BACKUP DATABASE sample
TABLESPACE ( syscatspace, userspace1, userspace2 )
ONLINE
TO /db2tbsp/backup1, /db2tbsp/backup2
```

上面例子中的第 2 行表明这是表空间备份，而不是完整数据库备份。还应注意，可以在备份中包括任意多个表空间，临时表空间不能使用表空间级备份进行备份。

通常，大家希望将相关的表空间备份在一起。例如，假设正在使用 DMS 表空间，其中一个表空间用于表数据，另一个表空间用于索引，还有一个表空间用于 LOB。那么应该同时备份这几个表空间，以便拥有一致的信息。对于所包含的表之间定义了约束的表空间而言，也应该如此。

7.3.3　增量备份

要执行增量备份，必须首先把 DB2 配置参数 trackmod 设置为 ON(修改完此参数后，数据库会被置为 backup pending 状态，此时需要对数据库执行完全备份才能继续访问数据库)，DB2 支持两种类型的增量备份：

- **增量备份：**DB2 备份自上一次数据库完整备份以来发生变化的所有数据。
- **delta 备份：**DB2 备份自上一次成功执行的完整备份、增量备份或 delta 备份以来发生变化的数据。

图 7-9 展示了这两种类型备份之间的不同之处。

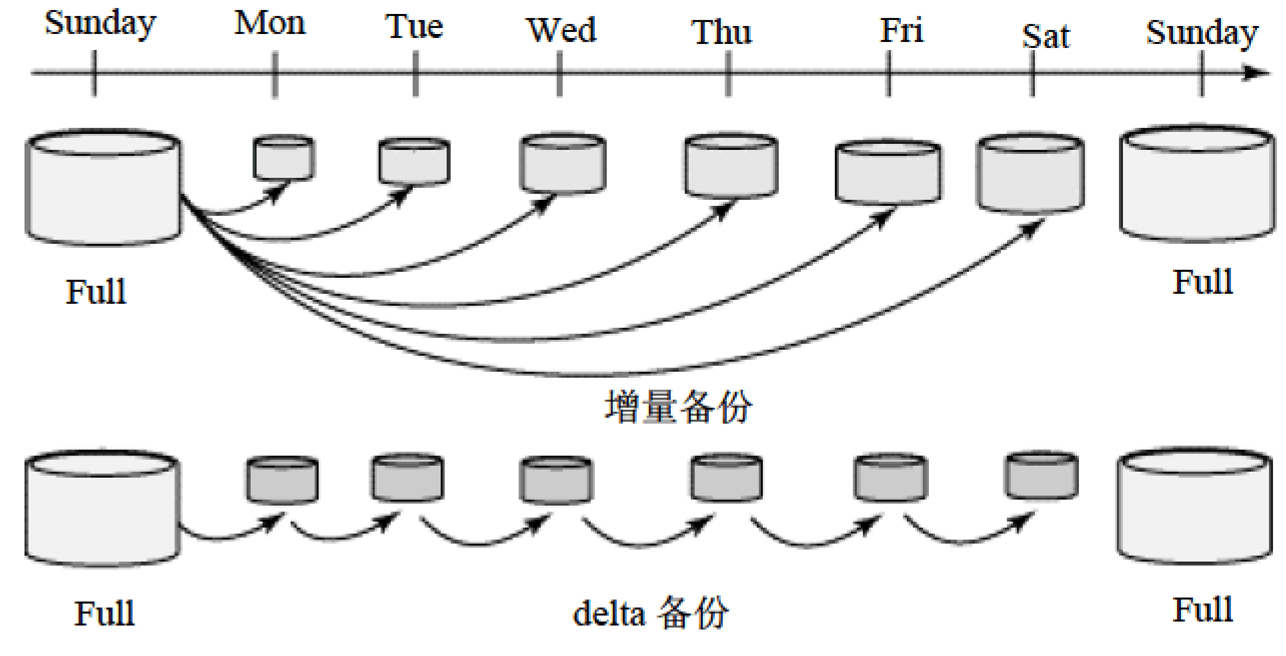


图 7-9　增量备份和 delta 备份之间的差异

在图 7-9 的顶部，在星期五执行了增量备份之后，如果系统出现崩溃，那么可以首先恢复第一个星期天的完整备份，然后再恢复在星期五做的增量备份。要执行增量(累积)备份，可以执行如下命令：

```
BACKUP DATABASE sample incremental TO /db2tbsp/backup1, /db2tbsp/backup2
```

其中，incremental 表示我们执行的是增量(累积)备份。

在图 7-9 的底部，在星期五执行了 delta 备份之后，如果系统出现崩溃，那么可以首先恢复第一个星期天的完整备份，然后再恢复从星期一到星期五做的每个 delta 备份。要执行 delta(差分)备份，可以执行如下命令：

```
BACKUP DATABASE sample incremental delta TO /db2tbsp/backup1, db2tbsp/backup2
```

其中，incremental delta 表示我们执行的是增量 delta(差分)备份。

使用控制中心执行备份

除了使用命令行外，我们还可以使用控制中心来执行备份。图 7-10 展示了如何从控制中心调用 BACKUP 实用程序。要执行数据库备份或表空间备份，可以在要备份的数据库上单击右键并选择“备份”，如图 7-10 所示。



图 7-10　选择要备份的数据库

接下来的图 7-11 展示了执行 BACKUP 实用程序所需的“备份向导”和选项。

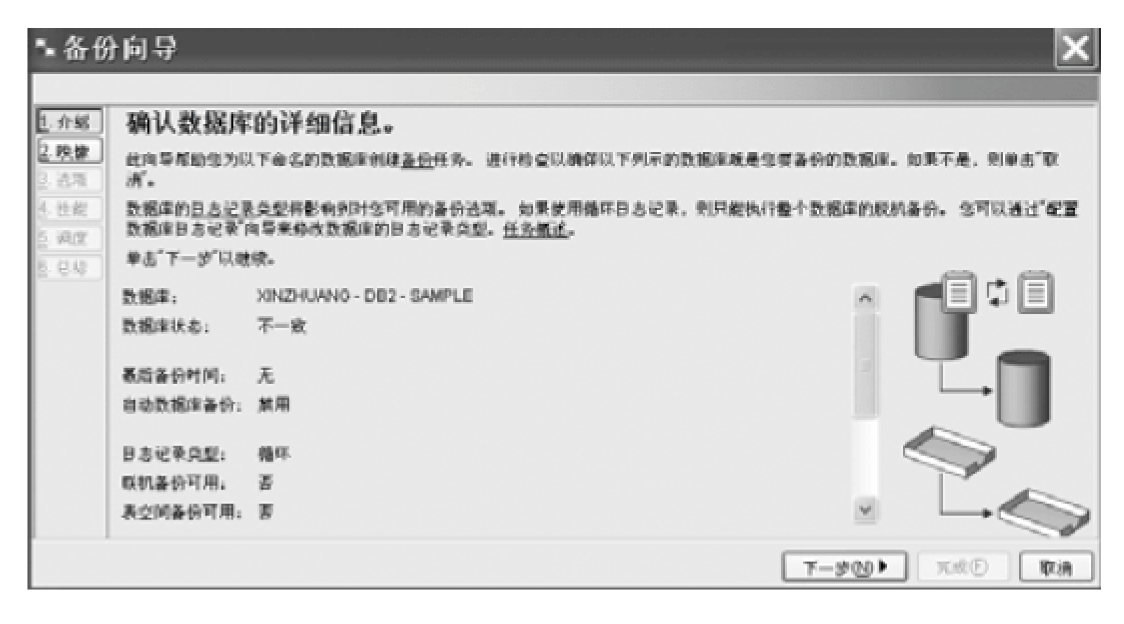


图 7-11　备份向导

备份文件的命名惯例

DB2 备份文件的命名惯例包括以下几项：

- 数据库别名
- 表示备份类型的数字(0 表示完整数据库备份，3 表示表空间备份，4 表示通过 LOAD 使用 COPY YES 选项创建的映像)
- 实例名称
- 数据库节点(对于单分区数据库，总是为 NODE0000)
- 编目节点号(对于单分区数据库，总是为 CATN0000)
- 备份的时间戳
- 映像序列号

图 7-12 展示了上述命名惯例，适用于所有平台。

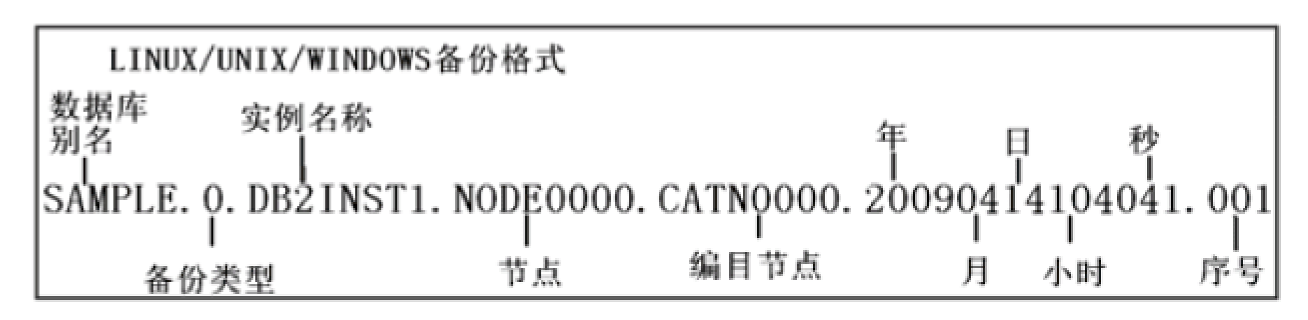


图 7-12　DB2 备份文件的命名惯例

7.3.4　检查备份完整性——db2ckbkp

当备份完成后，使用 db2ckbkp 命令不仅可以用来检查 DB2 数据库备份文件的完整性，而且还可以用来查询 DB2 数据库备份文件的元数据。如果有一些备份文件，但是不知道备份的类型，可以使用 db2ckbkp -h <备份文件>来检查 DB2 数据库备份的类型。

如果想检查一下文件 SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001 的备份类型，可以使用如下命令：

```
db2ckbkp -h SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
$db2ckbkp -h SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
=====================
MEDIA HEADER REACHED:
=====================
        Server Database Name           -- SAMPLE
        Server Database Alias          -- SAMPLE
        Client Database Alias          -- SAMPLE
        Timestamp                      -- 20081119232532
        Database Partition Number      -- 0
        Instance                       -- DB2
        Sequence Number                -- 1
```

```
        Release ID                          -- C00
        Database Seed                       -- 491AA028
        DB Comment's Codepage (Volume) -- 0
        DB Comment (Volume)                 --
        DB Comment's Codepage (System) -- 0
        DB Comment (System)                 --
        Authentication Value                -- 255
        Backup Mode                         -- 0
        Includes Logs                       -- 0
        Compression                         -- 0
        Backup Type                         -- 0
        Backup Gran                         -- 0
        Status Flags                        -- 11
        System Cats inc                     -- 1
        Catalog Partition Number            -- 0
        DB Codeset                          -- UTF-8
        DB Territory                        --
        LogID                               -- 1221200288
        LogPath                         --
  /db2/db2test/db2test1/NODE0000/SQL00002/SQLOGDIR/
        Backup Buffer Size                  -- 3411968
        Number of Sessions                  -- 1
        Platform                            -- 5
  The proper image file name would be:
  SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
  [1] Buffers processed:
############################################################
  Image Verification Complete - successful.
```

可以通过上述输出中的 Backup Mode、Backup Type 和 Backup Gran 来确定备份的类型，对 3 个关键字的说明如下：

- Backup Mode：0 表示 offline(脱机备份)，1 表示 online(联机备份)
- Backup Type：0 表示 full(全备份)，3 表示 tablespace(表空间备份)
- Backup Gran：0 表示 normal(正常备份)，16 表示 incremental(增量备份)，48 表示 delta(增量 delta 备份)

  其中，对 incremental(增量备份)和 delta(增量 delta 备份)的说明如下：

  ◊ incremental(增量备份)：增量备份是最近一次成功的完全备份时点之后的数据库操作的副本，因为每个增量备份都包含上一次增量备份的内容，所以也称为累积备份。最近一次成功的完全备份是增量备份的基础。

  ◊ delta(增量 delta 备份)：delta 备份映像或增量 delta 备份映像是自从上次数据库

的成功备份(包括完整、增量或 delta 备份)以来，已更改过的所有数据库数据的副本，也称为差异备份映像或非累积备份映像。delta 备份映像的前身是最新的成功备份，包括 delta 备份映像中的每个数据库备份。

- Includes Logs：表示在线备份期间是否包含备份期间产生的数据库日志。0 为否，1 为是。
- Compression：是否启用备份压缩。0 表示没有启动，1 表示启用备份压缩。

明确了上述数字的含义后，我们就可以很容易地辨别上述备份文件是否属于“联机全备份”：

- Backup Mode——1(联机备份)
- Backup Type——0(全备份)
- Backup Gran——0(正常备份)

如果要检查磁带上数据库备份的完整性，可以使用 db2ckbkp –n 选项。

7.4 数据库和表空间恢复

7.4.1 数据库恢复

本节讨论 RESTORE 实用程序，RESTORE 实用程序使用备份文件作为输入，输出是新的或已有的数据库。虽然我们讨论的是数据库恢复，但我们使用的实用程序是 RESTORE，而不是 RECOVERY。

为了恢复到已有的数据库，需要获得数据库的 SYSADM、SYSCTRL 或 SYSMAINT 权限。为了恢复到新的数据库，需要 SYSADM 或 SYSCTRL 权限。

下面是 RESTORE 命令的语法：

```
RESTORE DATABASE source-database-alias { restore-options | CONTINUE | ABORT }

restore-options:
  [USER username [USING password]]
  [Restore-Inventory-Clause] [INCREMENTAL [AUTOMATIC | ABORT]]
  [Media-Target-Clause] [TAKEN AT date-time]
  [[TO target-directory] | [ON path [{,path}...] [DBPATH ON path]]]
  [[TRANSPORT [STAGE IN staging-database-alias]] INTO target-database-alias]
  [LOGTARGET {directory | {{INCLUDE | EXCLUDE} [FORCE]}}]
  [NEWLOGPATH directory] [WITH num-buff BUFFERS] [BUFFER buffer-size]
  [REPLACE HISTORY FILE] [REPLACE EXISTING]
  [REDIRECT [GENERATE SCRIPT file-name]] [PARALLELISM n] [COMPRLIB lib-name]
  [COMPROPTS options-string] [WITHOUT ROLLING FORWARD]
```

```
    [WITHOUT PROMPTING]

  Restore-Inventory-Clause:
    rebuild-options |
    TABLESPACE [ONLINE] |
    TABLESPACE (tblspace-name [ {,tblspace-name} ... ])
    [SCHEMA (schema-name [ {,schema-name} ... ])] [ONLINE] |
    HISTORY FILE [ONLINE] |
    LOGS [ONLINE] |
    COMPRESSION LIBRARY [ONLINE]

  Media-Target-Clause:
    USE {TSM | XBSA} [OPEN num-sess SESSIONS] | SNAPSHOT [LIBRARY shared-lib]]
      [OPTIONS {options-string | options-filename}] |
    LOAD lib-name [OPEN num-sess SESSIONS]
      [OPTIONS {options-string | options-filename}] |
    FROM dir/dev [{,dir/dev} ... ]

  rebuild-options:
    REBUILD WITH {ALL TABLESPACES IN {DATABASE | IMAGE} [EXCEPT TABLESPACE
    (tblspace-name [ {,tblspace-name} ... ])] | TABLESPACE (tblspace-name
    [ {,tblspace-name} ... ])}
```

让我们看一个例子。要执行 sample 数据库恢复，可以使用以下命令：

```
  (1)  RESTORE DATABASE sample
  (2)  FROM  /DBBACKUP
  (3)  TAKEN AT 20080314131259
  (4)  WITHOUT ROLLING FORWARD
  (5)  WITHOUT PROMPTING
```

在上面的例子中：

(1) 表明要恢复的数据库映像的名称。

(2) 指定要从中读取输入备份文件的位置。

(3) 如果目录中有多个备份映像，该选项将基于时间戳(备份名称的一部分)标识特定的备份。

(4) 如果数据库启用了归档日志记录，那么当数据库被恢复时，将自动被置于 roll forward pending 状态。这一行告诉 DB2 不要将数据库置于 roll forward pending 状态。如果数据库被设置为 roll forward pending 状态，数据库将处于不访问的状态，这样就必须使用 roll forward 工具来前滚数据库，从而使数据库可以访问。

注意：

在下列情况中，不能使用 WITHOUT ROLLING FORWARD 选项：

- 从在线备份映像中复原
- 从表空间级的备份映像中复原

(5) 当执行 RESTORE 时，你将看不到任何提示。

注意，以上语法中没有关键字 OFFLINE，因为这是默认模式。对于 RESTORE 实用程序来说，这是能用于数据库的唯一模式。

如果备份映像中包括日志文件，那么可以使用 RESTORE DATABASE 命令的 LOGTARGET 选项恢复日志文件。

为了使用/DBBACKUP 目录中的备份映像恢复 sample 数据库，并将日志文件恢复到/DB2/SAMPLE/SQLOGDIR 目录，可以发出：

```
RESTORE DATABASE sample FROM /DBBACKUP
LOGTARGET /DB2/SAMPLE/SQLOGDIR
```

如果已经存在同名的数据库，在执行上面命令时会接收到如下警告：

```
    SQL2539W  Warning!  Restoring to an existing database that is the same as
the backup image database.  The database files will be deleted.
    Do you want to continue ? (y/n)
```

选择 y，开始恢复并将原有数据库删除。

如果只是需要备份映像中保留的日志文件，也可以通过使用 LOGS 关键字只恢复日志文件，而不恢复数据库：

```
RESTORE DATABASE sample LOGS FROM /DBBACKUP
LOGTARGET /DB2/SAMPLE/SQLOGDIR
```

7.4.2 表空间恢复

可以使用完整数据库备份或表空间备份来恢复表空间。对于表空间恢复，要小心地做好计划，因为一不小心就会使数据处于不一致状态。

下面是表空间 RESTORE 命令的一个例子：

```
(1)  RESTORE DATABASE sample
(2)  TABLESPACE ( mytblspace1 )
(3)  ONLINE
(4)  FROM /db2tbsp/backup1, /db2tbsp/backup2
```

在上面的例子中：

(1) 表明要恢复的数据库映像。

(2) 表明这是针对表空间的 RESTORE，并指定要恢复的表空间的名称。

(3) 表明这是在线恢复。对于用户表空间，在线恢复和离线恢复都是允许的。如前所述，对于整个数据库的恢复，只允许离线恢复。

(4) 指定输入备份文件所在的位置。

1. 关于表空间恢复的考虑

表空间被恢复之后，总是被置于 rollforward pending 状态。为了使表空间可以被访问并重置这个状态，必须至少将表空间前滚一个最小的时间点(minimum recovery time)，这个最小的时间点可以确保表空间和日志与系统编目表中的信息一致。

考虑下面这个例子：

(1) 假设在时间 t1 上做了完整数据库备份，其中包括表空间 mytbls1。

(2) 在时间 t2 上，你在表空间 mytbls1 中创建了表 myTable。这将为表空间 mytbs1 到 t2 的恢复设置最小时间点。

(3) 在时间 t3 上，你决定使用 t1 时做的完整数据库备份只恢复表空间 mytbls1。

(4) 恢复完成之后，表空间 mytbls1 将被置于 rollforward pending 状态。如果可以前滚到最小时间点之前的一个时间点上，那么表空间 mytbls1 中将不包括表 myTable；然而，系统编目却说这个表的确在 mytbls1 中。为了避免出现这样的不一致，当恢复表空间时，DB2 将强制使你至少前滚到最小时间点上。

当对表空间或表空间中的表运行 DDL 语句时，最小时间点将被更新。为了确定表空间恢复的最小时间点，可以使用以下两种方法之一：

- 使用 list tablespaces show detail 命令，示例如下：

```
$db2 list tablespaces show detail
………………..
Tablespace ID                        = 54
Name                                 = DB2TOOL
Type                                 = Database managed space
Contents                              = All permanent data. Large table space.
State                                = 0x0000
  Detailed explanation:
    Normal
Total pages                           = 262144
Useable pages                         = 262142
Used pages                           = 1192
Free pages                           = 260950
High water mark (pages)                = 1192
Page size (bytes)                     = 16384
Extent size (pages)                    = 2
```

```
  Prefetch size (pages)                 = 2
  Number of containers                  = 1
  Minimum recovery time                 = 2012-02-10-17.00.22.000000
 ……………..
```

● 使用 get snapshot for tablespace on <db_name>命令获得表空间快照，示例如下：

```
Tablespace name                            = DB2TOOL
  Tablespace ID                            = 54
  Tablespace Type                          = Database managed space
  Tablespace Content Type           = All permanent data. Large table space.
  Tablespace Page size (bytes)               = 16384
  ……………….
  ……………….
  Time of last successful resize              =
  Last resize attempt failed                = No
  Rebalancer Mode                          = No Rebalancing
  Minimum Recovery Time                      = 02/10/2012 17:00:22.000000
  Number of quiescers                        = 0
  Number of containers                       = 1
```

如果恢复的表空间中包含系统编目表空间(SYSCATSPACE)，那么必须将表空间前滚到日志的最后，并使它处于离线模式。在 7.5 节我们将更详细地讨论 ROLLFORWARD 命令。

2. 使用控制中心执行恢复

除了可以使用命令行恢复数据库和表空间外，我们还可以使用控制中心进行恢复。图 7-13 展示了如何从控制中心调用 RESTORE 实用程序。要执行数据库或表空间恢复，可以在要恢复的数据库上单击右键并选择“**复原**”。

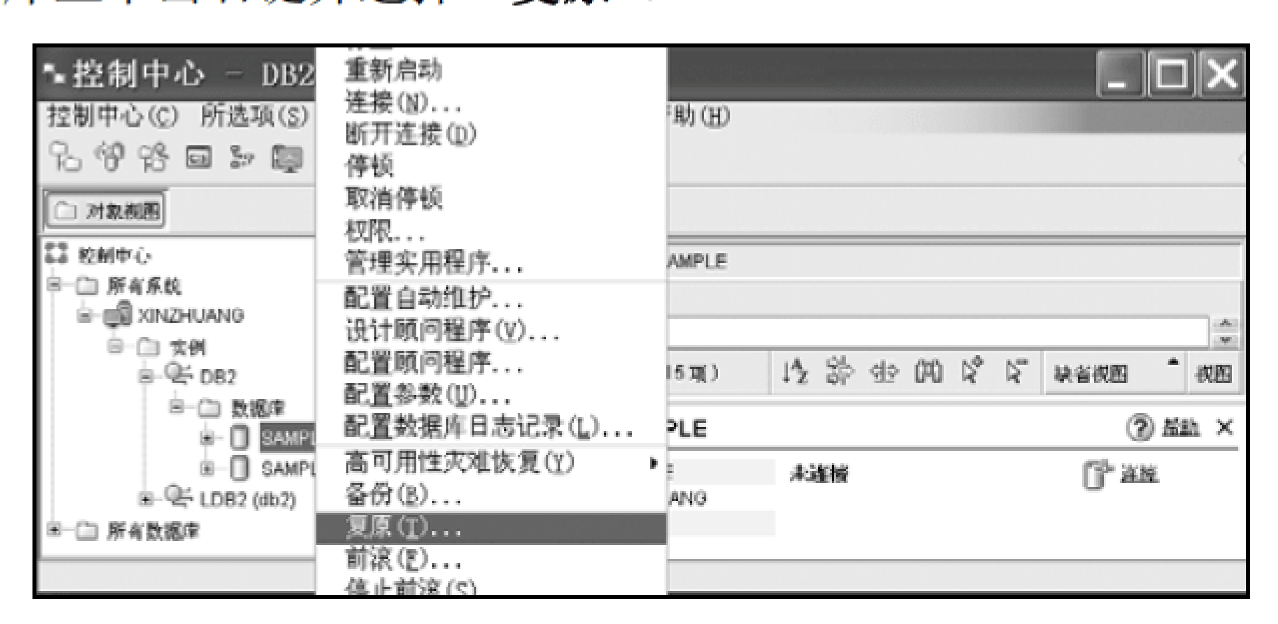


图 7-13　从控制中心调用 RESTORE 实用程序

接下来的图 7-14 展示了执行 RESTORE 实用程序所需的一些选项。

图 7-14　执行 RESTORE 实用程序所需的一些选项

7.4.3　增量恢复

增量备份是只包含自从执行前一个备份以来更新过的页的备份映像。要使用增量备份来执行恢复，可以在 RESTORE DATBASE 命令上指定 INCREMENTAL 选项。通常，增量恢复会涉及一系列的恢复操作。恢复中涉及的每个备份映像必须按以下顺序进行恢复：最近一次，第一次，第二次，第三次，依此类推，直到最后一个映像。

让我们看一个例子。下面是用于可恢复数据库 MYDB 的每周增量备份策略，其中包括一个每周完整数据库备份操作、一个每日差分(delta)备份操作和每周中的累积(增量)备份操作：

```
1. (Sun) backup db mydb to /backup
2. (Mon) backup db mydb online incremental delta to /backup
3. (Tue) backup db mydb online incremental delta to /backup
4. (Wed) backup db mydb online incremental to /backup
5. (Thu) backup db mydb online incremental delta to /backup
6. (Fri) backup db mydb online incremental delta to /backup
7. (Sat) backup db mydb online incremental to /backup
```

为了使用星期五创建的增量备份恢复数据库，需要发出一系列的 RESTORE DATABASE 命令来恢复所需的每个映像，顺序如下：

```
1. restore db mydb incremental taken at (Fri)----时间戳
2. restore db mydb incremental taken at (Sun)
3. restore db mydb incremental taken at (Wed)
4. restore db mydb incremental taken at (Thu)
5. restore db mydb incremental taken at (Fri)
```

可以看到，这个过程非常冗长，而且也容易出错。你必须知道要恢复的映像是什么，还要知道以什么顺序恢复这些映像。为了使这个过程变得更容易，可以使用自动增量恢复实用程序。为了执行自动增量恢复，只需要识别出要恢复的最近的备份映像，然后对它发出带 incremental automatic 选项的 RESTORE DATABASE 命令即可。恢复实用程序将负责剩下的事情。

上述所有手动恢复命令都可以放在单个自动增量恢复命令中：

```
restore db mydb incremental automatic taken at (Fri)
```

7.4.4 增量恢复检查——db2ckrst

对于 DB2 数据库来说，其非 delta 备份的恢复操作是可以通过仅发出两条恢复命令来完成的。但如果要恢复数据库的 delta 备份，可以使用 db2ckrst 这一检测 delta 备份恢复顺序的实用程序，检测要恢复的增量备份映像的时间戳，进而获取恢复操作的命令序列。下面是一个例子：

(1) 获取恢复操作的命令序列：

```
SAMPLE.0.DB2INST1.NODE0000.CATN0000.20080429143406.001;
 db2ckrst -d sample -t 20080429143406
```

(2) 命令执行后，在屏幕上会有如下类似输出：

```
Suggested restore order of images using timestamp 20080429143406 for
database sample.
=============================================
restore db sample incremental taken at 20080429143406
restore db sample incremental taken at 20080429142824
restore db sample incremental taken at 20080429143406
=============================================
```

(3) 恢复 delta 备份：

```
db2 restore db sample incremental taken at 20080429143406
db2 restore db sample incremental taken at 20080429142824
db2 restore db sample incremental taken at 20080429143406
```

7.4.5 重定向恢复

如果在需要恢复数据库的时候，容器的位置或名称与备份时的不同，那么又会如何呢？此时，使用默认的恢复方法会导致恢复失败。

重定向恢复可以解决以上问题。

数据库的重定向恢复通过使用带 REDIRECT 和 GENERATE SCRIPT 选项的 RESTORE 命令来生成重定向恢复脚本。当使用 GENERATE SCRIPT 选项时，恢复实用程序从备份映像中提取容器信息并生成 CLP 脚本，其中包括所有详细的容器信息。之后，可以在这个脚本中修改任何路径或容器大小，还可以运行 CLP 脚本，用一组新的容器重新创建数据库。

你已经看到了重定向恢复是如何工作的，重定向恢复还可用于为 SMS 表空间添加容器。我们在“第 3 章：创建数据库和表空间”中讲过，在大多数情况下，不能修改 SMS 表空间并为之添加容器。但是，重定向恢复可以绕过这个限制。

例如，要使用脚本来执行 sample 数据库的重定向恢复，可以遵循以下步骤：

(1) 使用恢复实用程序生成重定向恢复脚本：

```
    RESTORE DATABASE SAMPLE FROM /DBBACKUP INTO NEWDB REDIRECT GENERATE SCRIPT
newdb.clp WITHOUT ROLLING FORWARD
```

这会创建名为 *target-database-alias*.clp 的重定向恢复脚本，在这个例子中就是 newdb.clp。

(2) newdb.clp 包含执行重定向恢复所需的所有命令，此外还包含所有表空间信息。下面显示了 newdb.clp 脚本：

```
    ********************************************************************
    -- ** automatically created redirect restore script
    ********************************************************************
    UPDATE COMMAND OPTIONS USING S ON Z ON NEWDB NODE0000.out V ON;
    SET CLIENT ATTACH DBPARTITIONNUM  0;
    SET CLIENT CONNECT DBPARTITIONNUM 0;
    ********************************************************************
    -- ** automatically created redirect restore script
    ********************************************************************
    RESTORE DATABASE SAMPLE
    -- USER  username
    -- USING 'password'
    FROM '/dbbackup' TAKEN AT 20080516120102
    -- ON 'D:'
    -- DBPATH ON 'target-directory'
    INTO NEWDB
    -- NEWLOGPATH '/db2/db2test/db2test1/NODE0000/SQL00002/SQLOGDIR'
    -- WITH num-buff BUFFERS
    -- BUFFER buffer-size
    -- REPLACE HISTORY FILE
    -- REPLACE EXISTING REDIRECT
    -- PARALLELISM n WITHOUT ROLLING FORWARD
```

```
-- WITHOUT PROMPTING;
********************************************************************
-- ** table space definition
********************************************************************
-- ** Tablespace name                          = SYSCATSPACE
-- **   Tablespace ID                          = 0
-- **   Tablespace Type                        = Database managed space
-- **   Tablespace Content Type                = All permanent data. Regular
                                                 table space.
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 4
-- **   Using automatic storage                = Yes
-- **   Auto-resize enabled                    = Yes
-- **   Total number of pages                  = 16384
-- **   Number of usable pages                 = 16380
-- **   High water mark (pages)                = 8872
********************************************************************
-- ** Tablespace name                          = TEMPSPACE1
-- **   Tablespace ID                          = 1
-- **   Tablespace Type                        = System managed space
-- **   Tablespace Content Type                = System Temporary data
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = Yes
-- **   Total number of pages                  = 1
********************************************************************
-- ** Tablespace name                          = USERSPACE1
-- **   Tablespace ID                          = 2
-- **   Tablespace Type                        = Database managed space
-- **   Tablespace Content Type                = All permanent data. Large
                                                 table space.
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = Yes
-- **   Auto-resize enabled                    = Yes
-- **   Total number of pages                  = 8192
-- **   Number of usable pages                 = 8160
-- **   High water mark (pages)                = 1888
********************************************************************
-- ** Tablespace name                          = TBS1
-- **   Tablespace ID                          = 5
-- **   Tablespace Type                        = Database managed space
-- **   Tablespace Content Type          = All permanent data. Large table  space
```

```
-- **   Tablespace Page size (bytes)           = 4096
-- **   Tablespace Extent size (pages)         = 32
-- **   Using automatic storage                = No
-- **   Auto-resize enabled                    = No
-- **   Total number of pages                  = 1000
-- **   Number of usable pages                 = 960
-- **   High water mark (pages)                = 96
********************************************************************
SET TABLESPACE CONTAINERS FOR 5
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (FILE   '/DB2/SAMPLE/tbs1'  1000);
********************************************************************
-- ** start redirected restore
********************************************************************
RESTORE DATABASE NEWDB CONTINUE;
********************************************************************
-- ** end of file
********************************************************************
```

'--'表明是注释。SET TABLESPACE CONTAINER 命令只是用于没有使用自动存储的表空间。对于使用自动存储的表空间，它们的容器是由 DB2 自动处理的，因此不需要重新设置它们。

(3) 通过在文本编辑器中打开重定向恢复脚本，可以对其进行修改。可以修改：

- 恢复选项
- 容器布局、路径或大小，但表空间容器的大小不能小于备份映像中保存的高水位

(4) 运行修改后的重定向恢复脚本：

```
db2 -tvf newdb.clp
```

这个脚本的输出将被写到名为 *databasename_nodenumber*.out 的文件中。

7.4.6　恢复已删除的表

你可能会意外删除(drop)仍然需要的表。为了能够在这种情况下恢复被意外删除的表，你应该考虑事先将表设置为可复原的。已删除的表的恢复功能使得可以使用表空间级的复原和前滚操作来恢复已删除的表数据。这样会比数据库级的恢复要快，而且数据库中的其他对象将对用户保持可用。

要使已删除的表可以复原，必须对表所在的表空间启用 DROPPED TABLE RECOVERY 选项。这可以在表空间创建期间或者通过调用 ALTER TABLESPACE 语句来完成，这个选项在 DB2 V9 中默认是启用的。DROPPED TABLE RECOVERY 选项是表空间的属性，并

限于对常规表空间使用。要确定是否对表空间启用了已删除表的恢复功能，可以查询 SYSCAT.TABLESPACES 目录表中的 DROP_RECOVERY 列。

对表(已对表的表空间启用了 DROPPED TABLE RECOVERY 选项)运行 DROP TABLE 语句时，将在日志文件中建立特定的条目用于标识已删除的表，还会在历史文件(db2rhist.asc)中记录包含可用于重新创建表的信息。如果存在大量的删除表的操作，那么历史文件的大小也会随着增大，而历史文件的大小会在多种情况下影响数据库的性能。

创建表空间时，默认情况下已删除的表的恢复选项已打开。如果不想启用这个功能，那么可以在发出 CREATE TABLESPACE 命令时显式将 DROPPED TABLE RECOVERY 选项设置为 OFF，也可以使用 ALTER TABLESPACE 命令对现有表空间禁用这个功能。如果有许多删除表的操作，那么可能会由于要恢复的内容较多或者历史记录文件很大而对性能造成负面影响。这是在考虑这个选项时需要参考的因素，这就需要我们仔细评估一些表空间中保存的表是否必须设置这个选项，比如一些表空间保存的都是一些临时性的表，会经常出现删除表的操作，而且这些临时表中数据的丢失也可以接受，此时就可以考虑将这个表空间的 DROPPED TABLE RECOVERY 选项设置为 OFF。

使用 DROPPED TABLE RECOVERY 选项时有下面的限制：

- DROPPED TABLE RECOVERY 选项不能用于系统临时表空间(System Temporary Tablespace)。
- 大对象(LOB)或长字段数据。对于大型表空间，不支持 DROPPED TABLE RECOVERY 选项。如果尝试复原包含 LOB 或 LONG VARCHAR 列的已删除表，这些列将在生成的导出文件中设置为 NULL。仅可对常规表空间使用 DROPPED TABLE RECOVERY 选项，而不能对临时或大型表空间使用。
- XML 数据。如果尝试恢复包含 XML 数据的已删除表，那么相应的列数据将为空。

如果表在删除时处于重组暂挂状态，那么历史记录文件中的 CREATE TABLE DDL 与导入文件中的 CREATE TABLE DDL 不完全匹配。在执行重组建议的第一个 ALTER 操作之前，导入文件将采用表的格式，但历史记录文件中的 CREATE TABLE 语句将与包含任何 ALTER TABLE 语句结果的表的状态相匹配。

如果正在恢复 GRAPHIC 或 VARGRAPHIC 数据类型的数据，那么数据可能会包括多个代码页。为恢复数据，需要指定 IMPORT 或 LOAD 命令的 USEGRAPHICCODEPAGE 文件类型修饰符。在这种情况下，使用 LOAD 命令来恢复数据将会提高恢复操作的性能。

一次只能复原一个已删除表。可执行下列操作来复原已删除表：

(1) 通过调用 LIST HISTORY DROPPED TABLE 命令来标识已删除表。已删除表的标识列为“备份标识”列。

(2) 复原在删除表之前建立的数据库级或表空间级备份映像。

(3) 创建包含表数据的文件将写至的导出目录。此目录必须可供所有数据库分区访问，或者在每个数据库分区上都存在。导出目录下的子目录是由每个数据库分区自动创建的，这些子目录的名称是 NODEnnnn，其中 nnnn 代表数据库分区或节点号。包含已删除表数据的数据文件(就如同存在于每个数据库分区上那样)将导出到称为 data 的较低子目录中，例如\export_directory\NODE0000\data。

(4) 在删除表之后前滚至某时间点，对 ROLLFORWARD DATABASE 命令使用 RECOVER DROPPED TABLE 选项。也可前滚至日志末尾，以使对表空间或数据库中其他表进行的更新不会丢失。

(5) 使用 CREATE TABLE 语句从历史文件重新创建表。

(6) 将在前滚操作期间导出的表数据导入表中。如果表在删除时处于重组暂挂状态，那么需要更改 CREATE TABLE DDL 的内容以与数据文件的内容相匹配。

下面举例来说明。

(1) 执行完全数据库备份，需要注意备份映像的时间戳。

(2) 连接到数据库并创建表，执行生成日志记录的操作，插入几条记录：

```
    CONNECT TO sample
       CREATE TABLE tab1(no INTEGER) IN ts1
       INSERT INTO tab1 VALUES(1),(2),(3),(4),(5)
       ARCHIVE LOG FOR DATABASE sample
           SELECT * FROM tab1 /* check the 5 committed values from TAB */
```

(3) 模拟意外删除表的场景：

```
    DROP TABLE tab1
       COMMIT
       SELECT * FROM tab1
```

将返回以下错误消息：

```
    Error: SQL0204N "Administrator.TAB1" is an undefined name
```

(4) 恢复数据库。

要恢复已删除表，需要先恢复数据库备份，然后执行向前恢复(rollforward)操作：

```
    RESTORE DATABASE sample FROM c:\backup TAKEN AT 20080314144204 INTO sample
```

将返回以下消息：

```
    SQL2539W Warning! Restoring to an existing database that is the same as the
Backup image database.
       The database files will be deleted.
```

```
        Do you want to continue? (Y/N)        按 Y 键完成此过程
```

(5) 检索已删除表的对象 ID。

使用以下命令检索意外删除的表的对象 ID：

```
  LIST HISTORY DROPPED TABLE ALL FOR DATABASE sample
```

可以将返回的信息(例如表 7-3 中显示的示例)复制到某个文本文件中以供未来引用。

表 7-3　LIST HISTORY 命令返回的信息

```
Op Obi Timestamp Sequence Type Dev Earliest Log Current Log   Backup ID
-- --- ------------------ ---- --- ------------ ------------ ----------------------------
D  T  20080314142913                         000000000000892700050108
-------------------------------------------------------------------------------------------
"ADMINISTRATOR"."TAB1" resides in 1 table space(s):
00001 TS1
----------------------------------------------------------------------------
Comment: DROP TABLE
Start Time: 20080314142913
End Time: 20080314142913
Status: A
----------------------------------------------------------------------------
EID: 37
DDL: CREATE TABLE "ADMINISTRATOR"."TAB1" ( "NO" INTEGER )  IN "TS1" ;
```

表 7-3 中的 Backup ID 栏显示被删除的表的 ID 为 000000000000892700050108。这一信息对于恢复表非常重要。

(6) 向前恢复数据库。

现在已经获得了被删除的表的 ID，下一步需要使用表的备份 ID 恢复数据库，这样才能够导入表的数据。在向前恢复数据库之前，需要确保有目录可供存储导入数据，比如 c:\sample\exporttab1。使用以下命令向前恢复数据库：

```
   ROLLFORWARD DATABASE sample TO END OF LOGS    AND STOP RECOVER DROPPED TABLE
000000000000892700050108 TO c:\sample\exporttab1
```

END OF LOGS 选项的作用是让 DB2 在执行备份操作后应用所有可用日记文件。

(7) 检查导入的数据文件。

完成数据库向前恢复之后，需要检查在 ROLLFORWARD 命令中指定的路径。应该能

够找到一个. txt 文件，打开该文件并验证其中包含的数据与意外删除的表之前的数据是否相同。

(8) 连接到数据库并重新创建被删除的表。

验证导出文件之后，我们需要重新创建被删除的表并填入数据。被删除的表的定义包含在步骤(5)的 LIST HISTORY 命令的输出中。连接到数据库并执行 CREATE TABLE 语句：

```
    CONNECT TO sample
       CREATE TABLE "ADMINISTRATOR"."TAB1" ( "NO" INTEGER ) IN "TS1"
```

(9) 导入数据。

重新创建表之后，可以使用以下命令将数据库重新导入到表中：

```
    IMPORT FROM c:\sample\exporttab1\Node0000\data.txt OF DEL INSERT INTO
administrator.tab1
```

IMPORT 工具将导出文件中的所有数据导回到表中，并在成功后发送报告(未显示)。

(10) 验证恢复后的数据。

确保在 IMPORT 过程中没有错误或报警，并且所有数据都已导回表中：

```
    SELECT * FROM tab1
```

如果一切运行正常，那么在意外删除时间点之前的所有数据应该都在表中。

7.5　数据库和表空间前滚

7.5.1　数据库前滚

ROLLFORWARD 命令允许将数据库恢复至某个时间点，该命令是通过读取日志中相应的数据记录来完成这一过程的。虽然可以将数据库或表空间前滚到最小时间点之后的任何时间点，但是不能保证前滚到那个时间后所有数据都处于一致状态。

虽然本节不讨论 QUIESCE 命令，但是值得一提的是，可以使用这个命令在数据库正常操作期间设置一致点。通过设置这些一致点，总可以使时间点恢复到任何一致点，从而保证数据同步。但是需要注意的是：此命令会导致表空间处于不可访问的状态，因此需要仔细评估后才可使用。

一致点和很多其他信息(如备份、恢复、LOAD 等等)一起存储在 DB2 历史文件中，这个文件的内容可以使用 LIST HISTORY 命令来查看。

在前滚处理期间，DB2 将：

- 在当前日志路径中查找所需的日志文件。
- 如果找到日志文件，就应用(redo)日志文件中的事务。
- 如果在当前日志路径中没有发现那个日志文件，就搜索 OVERFLOW LOG PATH 选项指定的路径，并使用那个位置的日志文件。
- 如果在当前路径中没有发现那个日志文件，并且没有使用 OVERFLOW LOG PATH 选项，就使用 LOGARCHMETH1 中指定的方法从归档中获取日志文件。
- 如果日志在当前日志路径或 OVERFLOW LOG PATH 中，就重新应用事务。

要执行 ROLLFORWARD 命令，需要有 SYSADM、SYSCTRL 或 SYSMAINT 权限。

ROLLFORWARD 命令的语法如下：

```
ROLLFORWARD DATABASE database-alias [USER username [USING password]]
[TO {isotime [ON ALL DBPARTITIONNUMS] [USING LOCAL TIME | USING UTC TIME]} |
    {END OF BACKUP [ON ALL DBPARTITIONNUMS]} |
    {END OF LOGS [On-DbPartitionNum-Clause]}
    [AND {COMPLETE | STOP}]] |
[COMPLETE | STOP | CANCEL | {QUERY STATUS [USING LOCAL TIME | USING UTC TIME]}
    [On-DbPartitionNum-Clause]]
[TABLESPACE ONLINE |
   TABLESPACE (tblspace-name [ {,tblspace-name} ... ]) [ONLINE]]
[OVERFLOW LOG PATH (log-directory
   [{,log-directory ON DBPARTITIONNUM db-partition-number} ... ])]
[NORETRIEVE]
[RECOVER DROPPED TABLE dropped-table-id TO export-directory]

On-DbPartitionNum-Clause:
  ON {{DBPARTITIONNUM | DBPARTITIONNUMS} (db-partition-number
  [TO  db-partition-number] , ... ) | ALL DBPARTITIONNUMS [EXCEPT
  {DBPARTITIONNUM | DBPARTITIONNUMS} (db-partition-number
  [TO db-partition-number] , ...)]}
```

我们来看一个例子。要执行 sample 数据库的前滚操作，可以使用以下任何语句：

```
(1)ROLLFORWARD DATABASE sample TO END OF LOGS AND COMPLETE
(2)ROLLFORWARD DATABASE sample TO timestamp AND COMPLETE
(3)ROLLFORWARD DATABASE sample TO timestamp USING LOCAL TIME AND COMPLETE
```

从上面的代码中：

(1) 在这个例子中，我们将前滚到日志的最后，这意味着所有归档的日志和活动日志都将被遍历。最后，sample 数据库将完成前滚，并通过回滚任何未提交的事务来脱离 rollforward-pending 状态。

(2) 对于这个例子，DB2 将前滚到指定的时间点。在此语法中，时间必须是 UTC 时间。

(3) 这个例子类似于前一个例子，但是时间戳可以用当地时间表示。

语法中没有关键字 OFFLINE，因为这是默认模式。对于 ROLLFORWARD 命令，这是可用于数据库前滚的唯一模式。

7.5.2　表空间前滚

表空间前滚通常可以在线进行或离线进行。一个例外就是系统编目表空间(SYSCATSPACE)，这种表空间只能离线前滚。

下面是表空间前滚的示例：

```
ROLLFORWARD DATABASE sample   TO END OF LOGS AND COMPLETE
  TABLESPACE ( userspace1 ) ONLINE
```

在 7.5.1 节中，我们解释了这个例子中的选项。这里唯一没有解释过的是 TABLESPACE 选项，该选项指定要前滚的表空间。

1. 关于表空间前滚的考虑

如果启用注册变量 DB2_COLLECT_TS_REC_INFO，那么只需要处理恢复表空间所需的日志文件。ROLLFORWARD 命令将忽略不需要的日志文件，这样可以缩短恢复时间。

ROLLFORWARD 命令的 QUERY STATUS 选项可用于列出：

- DB2 已经前滚的日志文件
- 需要的下一个归档日志文件
- 自前滚处理开始以来最近提交的事务的时间戳

例如：

```
ROLLFORWARD DATABASE sample QUERY STATUS USING LOCAL TIME
```

在某个表空间时间点前滚操作完成之后，表空间被置于 backup-pending 状态。这里强制使用表空间或数据库的备份。

2. 使用控制中心执行前滚操作

除了可以使用命令行前滚数据库以外，还可以通过控制中心执行前滚操作。图 7-15 展示了如何从控制中心调用 ROLLFORWARD 命令。要执行数据库前滚或表空间前滚，可以在要前滚的数据库上单击右键并选择“**前滚**”。

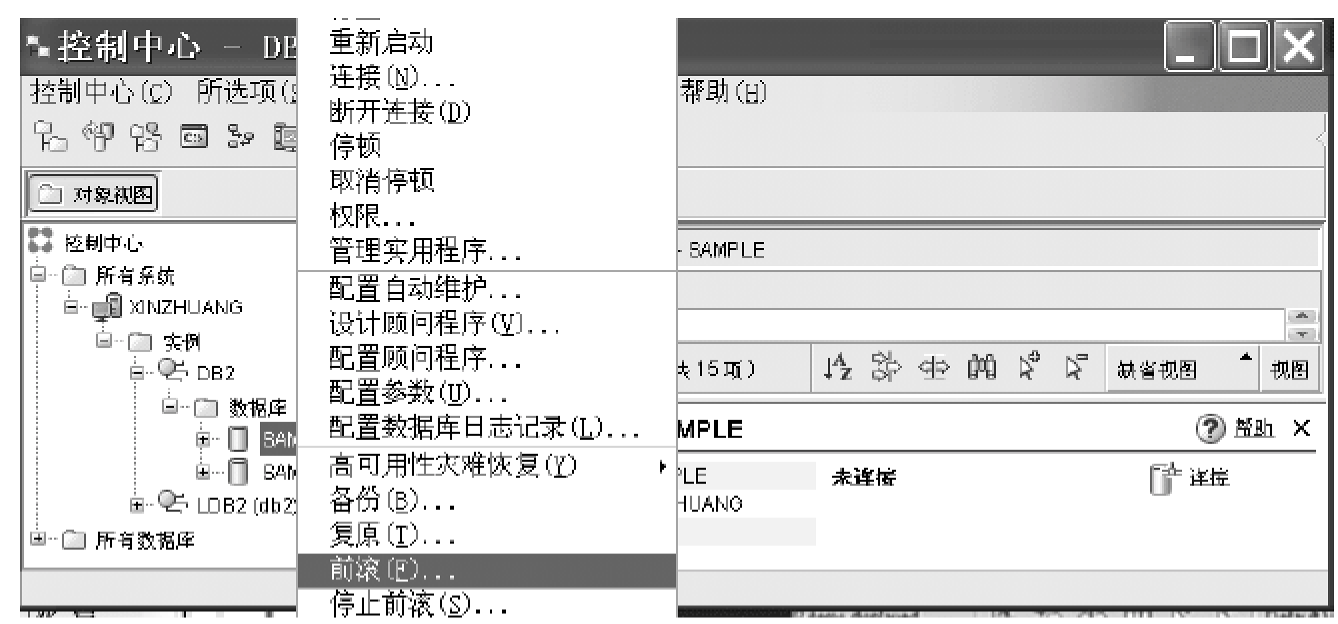


图 7-15　从控制中心调用 ROLLFORWARD 命令

图 7-16 展示了执行 ROLLFORWARD 命令所需的一些选项。

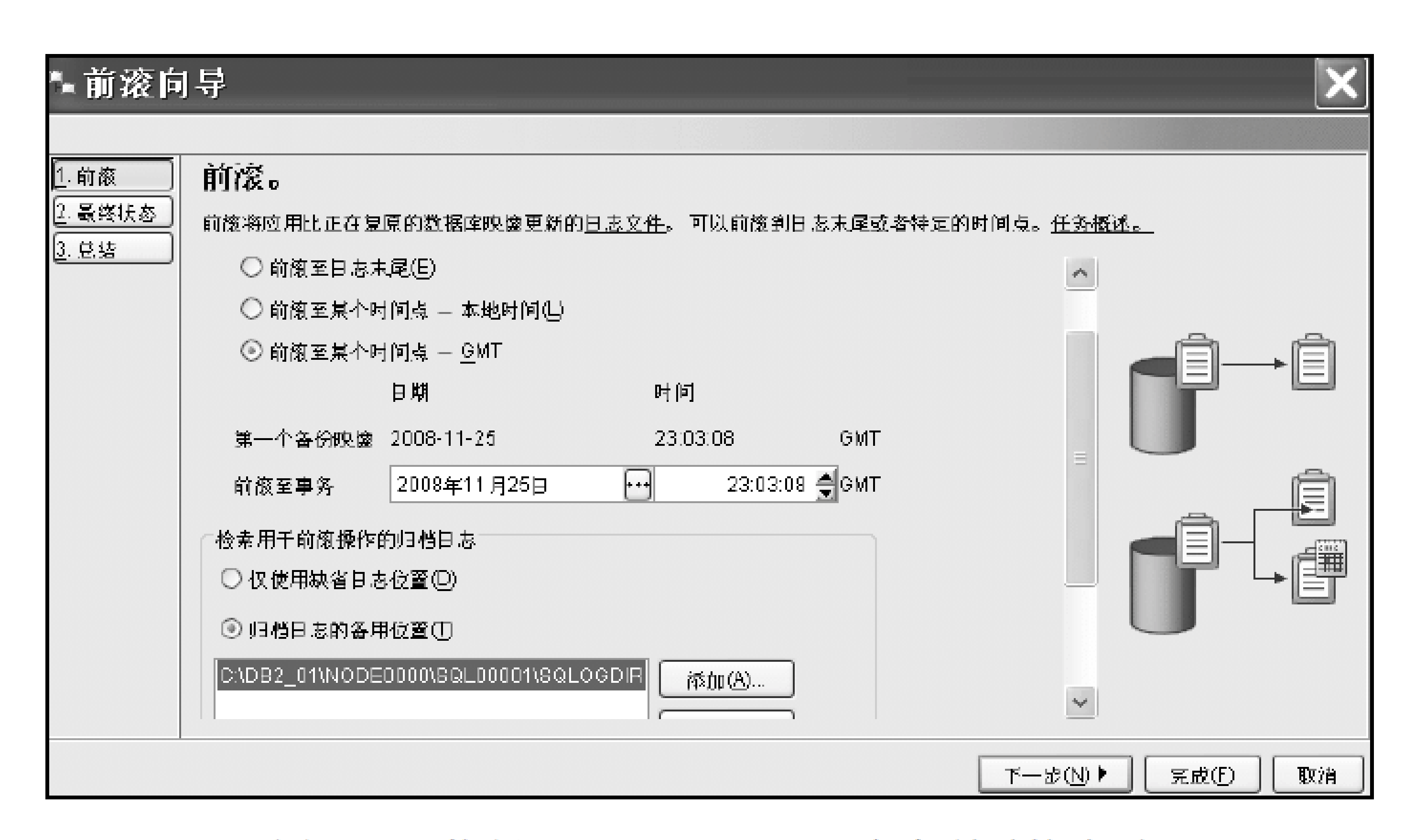


图 7-16　执行 ROLLFORWARD 命令所需的选项

3. 恢复示例

到目前为止，我们讲解了 BACKUP、RESTORE 和 ROLLFORWARD 命令。接下来我们通过列举一些场景来帮助理解不同类型的恢复。

对于图 7-17 所示的这个场景，使用了循环日志模式：

在 t6，数据库出现了一次计划外的停电事故。在 t7，DB2 被重新启动，当连接到数据库时，崩溃恢复(Crash Recovery)自动启动(假设数据库的 AUTORESTART 为 ON，此为默认值；否则，需要使用 RESTART DATABASE 命令手动启动)。崩溃恢复将遍历活动日志，并重新执行提交的事务。如果一个事务还没有被提交，它将被回滚。对于这个例子，两个插入操作将重新执行，而删除语句将被撤销。

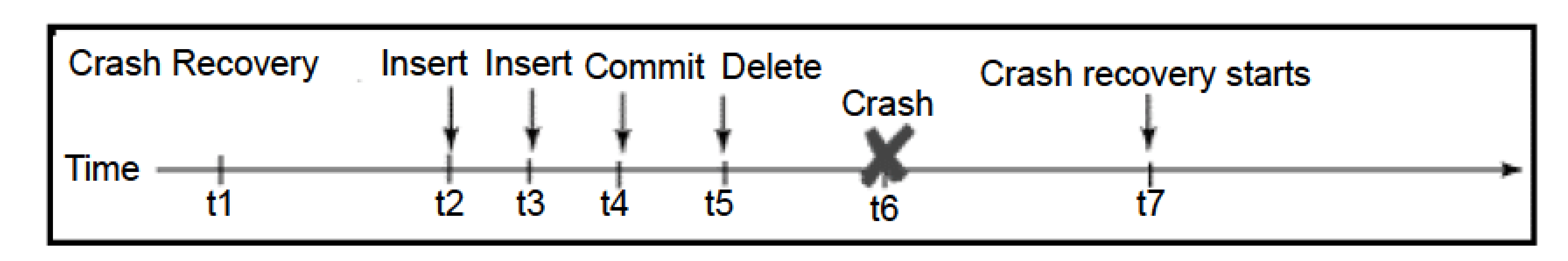


图 7-17　场景 1

对于图 7-18 所示的这个场景，循环日志记录在生效：

在 t7，你意识到所有表空间中的数据已经被 t6 时启动的某个事务毁坏。在 t8，你决定使用 t1 时做的完整数据库备份进行恢复。由于循环日志记录在生效，日志中很多已提交和写到硬盘上的事务已经被覆盖，因此不能应用日志(不能在循环日志记录中运行 ROLLFORWARD 命令，甚至不能前滚活动日志)。结果是：t2 至 t4 这段时间内很多好的事务将丢失。

对于图 7-19 所示的这个场景，使用了归档日志：

这是对场景 2 的扩展。在这个例子中，所有的日志已经被保存(归档日志)；在 t8 应用完整数据库恢复之后，可以将日志前滚到 t9。日志可以从 t1 前滚到任意时间点，但是你很可能不想超过 t6，因为在 t6 开始出现有故障的事务。

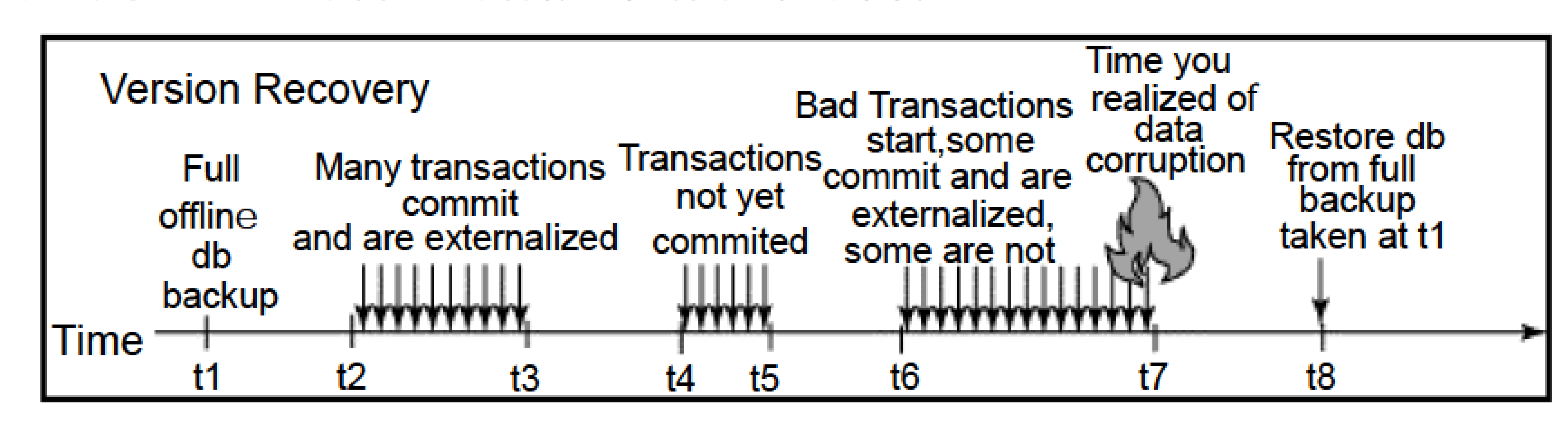


图 7-18　场景 2

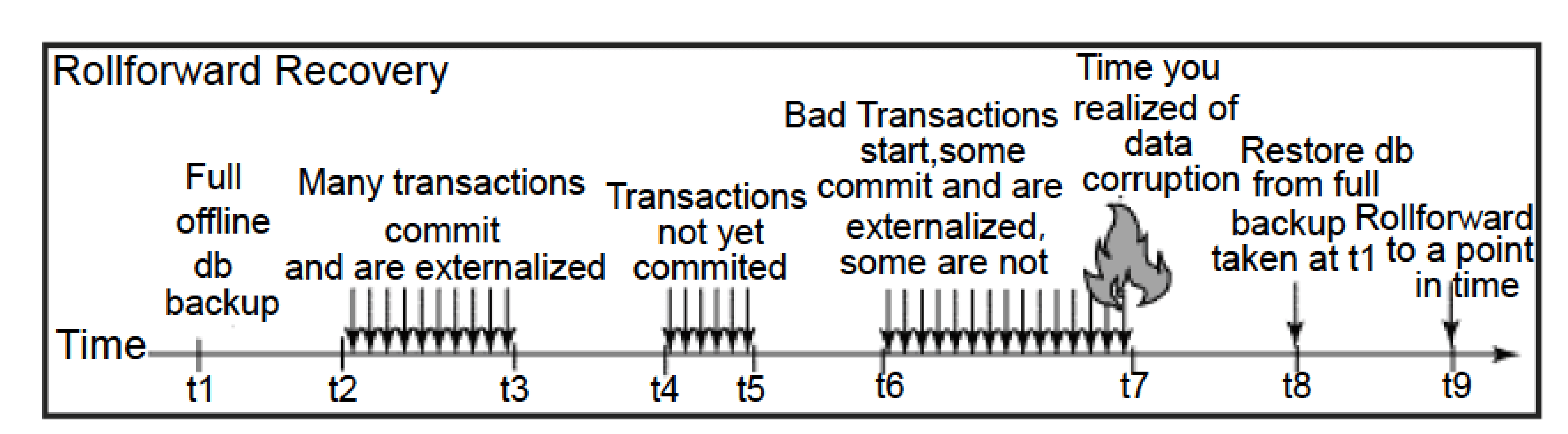


图 7-19　场景 3

图 7-20 所示的场景更详细地回顾了所有这些概念。

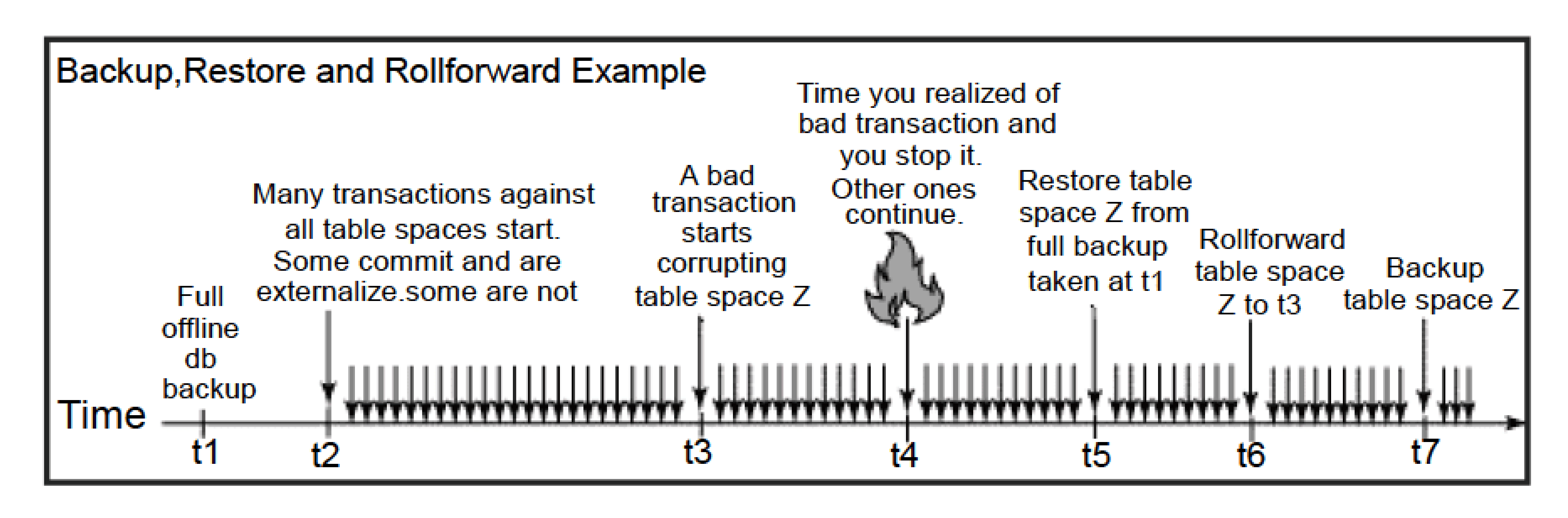


图 7-20　场景 4

t1 时完成了一次离线数据库备份。

t2 时执行日常事务。

在 t4，你意识到某个事务已经毁坏了表空间 Z，并且停止这个事务。针对其他表空间的其他事务则继续。

只是对于表空间 Z，你希望数据库处于 t3 之前的状态(开始坏事务之前的状态)，因此：

- 在 t5，使用 t1 做的完整离线备份恢复表空间 Z。
- 完成恢复后，表空间将处于 rollforward pending 状态。
- 在 t6，将这个表空间 Z 前滚到 t3，即故障事务开始之前。
- 你已经执行了一次时间点恢复。因此，DB2 现在将由于一致性的原因而使表空间处于 backup pending 状态。
- 在 t7，备份表空间。这时，数据库是一致的，所有用户和应用程序可以正常工作。被恢复的表空间在 t3 到 t7 之间将存在间隔，这就是我们的目标—— 删除被毁坏的数据。

7.6　RECOVER 实用程序

DB2 从 V8.2 开始提供了新命令 RECOVER DATABASE，新的 RECOVER DATABASE 命令结合了 RESTORE DATABASE 和 ROLLFORWARD DATABASE 命令的功能。过去我们在恢复数据库时先使用 RESTORE 命令把数据库恢复到备份的那个时刻，然后再使用 ROLLFORWARD 前滚(redo)日志中的 SQL 操作。那么能不能把这两个命令结合到一起呢？这就是 RECOVER 实用程序的由来，简单来说，recover=restore+rollforward。所以你如果仍然习惯过去那种传统的恢复方式(restore+rollforward)，那么可以不用这个实用程序。使用此命令时，根据恢复历史文件中的信息使数据库恢复到指定的时间，自动选择最佳适用

备份映像来执行恢复操作。要指定想要数据库恢复到的时间点，不需要指示必须复原哪个数据库备份映像或需要哪些日志文件以达到指定的时间点。RECOVER DATABASE 命令还支持恢复到日志文件末尾的操作。

RECOVER DATABASE 命令的语法是：

```
RECOVER DATABASE source-database-alias TO isotime [USING LOCAL TIME]
[USER username [USING password]
[USING HISTORY FILE history-file]
[OVERFLOW LOG PATH directory]
[RESTART]
```

例 7-1　在开发服务器上有一个 sample 数据库。昨夜的一次停电事故导致数据库中的数据遭到毁坏。你需要尽快恢复数据库。首先可以找到适当的备份，然后找到所需的 DB2 日志，以便前滚到停电事故之前的时间点。恢复 sample 数据库的一种更容易的方法是发出 RECOVER DB 命令，如下所示：

```
RECOVER DB sample
RECOVER DB sample TO 2008-05-21-13.50.00 USING LOCAL TIME
```

在第 1 行，DB2 从可用的最近备份映像恢复 sample 数据库，并且将前滚到日志的最后。在第 2 行，DB2 将 sample 数据库恢复到时间点 2008-05-21-13.50.00，这是按本地时间指定的。

RECOVER DATABASE 实用程序依赖于数据库恢复历史文件来发现最适合用于恢复的备份映像。在上面的恢复命令中，我们没有指定历史文件的位置。由于 sample 数据库已经在服务器上，因而 DB2 可以在数据库目录路径下找到历史文件。

如果要恢复的数据库不存在，那么必须指定历史文件的位置：

```
RECOVER DB sample TO END OF LOGS USING HISTORY FILE
 (/home/user/oldfiles/db2rhist.asc)
```

在用于存放恢复后的数据库的服务器上必须存在有效的历史文件，其中包含所需的备份映像和日志。在上面的命令中，如果没有可用的历史文件(由文件传输或历史文件备份得到)，那么在运行 RECOVER DATABASE 命令之前，就必须设法从备份映像中提取出历史文件。在这种情况下，连续运行标准的 RESTORE 和 ROLL FORWARD 命令来恢复数据库也许更容易一些。

例 7-2　拥有数据库 sample 的多个备份：

```
C:\>db2 list history backup all for db sample
列示 sample 的历史文件
```

```
匹配的文件条目数 = 4
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025222808001 F D S0000000.LOG S0000000.LOG
------------------------------------------------------------------------
包含 2 表空间：
00001 SYSCATSPACE
00002 USERSPACE1
------------------------------------------------------------------------
注释：DB2 BACKUP SAMPLE OFFLINE
开始时间：20081025222808
结束时间：20081025222829
状态：A
------------------------------------------------------------------------
EID：1 位置：C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025223024001 N D S0000001.LOG S0000001.LOG
------------------------------------------------------------------------
包含 2 表空间：
00001 SYSCATSPACE
00002 USERSPACE1
------------------------------------------------------------------------
注释：DB2 BACKUP SAMPLE ONLINE
开始时间：20081025223024
结束时间：20081025223045
状态：A
------------------------------------------------------------------------
EID：2 位置：C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025223239001 O D S0000003.LOG S0000003.LOG
------------------------------------------------------------------------
包含 2 表空间：
00001 SYSCATSPACE
00002 USERSPACE1
------------------------------------------------------------------------
注释：DB2 BACKUP SAMPLE ONLINE
开始时间：20081025223239
结束时间：20081025223300
状态：A
------------------------------------------------------------------------
EID：3 位置：C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
```

```
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025223408001 D D S0000005.LOG S0000005.LOG
-------------------------------------------------------------------------
包含 2 表空间:
00001 SYSCATSPACE
00002 USERSPACE1
-------------------------------------------------------------------------
注释: DB2 BACKUP SAMPLE OFFLINE
开始时间: 20081025223408
结束时间: 20081025223428
状态: A
-------------------------------------------------------------------------
EID: 4 位置: C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
```

如果想把数据库恢复到时间点 2008 年 10 月 25 日 22 点 30 分，只需要运行如下命令就可以了：

```
C:\>db2 recover database sample to 2008-10-25-22.30.00.000000
前滚状态
输入数据库别名 = sample
节点数已返回状态 = 1
节点号 = 0
前滚状态 = 未暂挂
下一个要读取的日志文件 =
已处理的日志文件 = S0000000.LOG - S0000000.LOG
上次落实的事务 = 2008-10-25-22.30.00.000000
DB20000I RECOVER DATABASE 命令成功完成
```

另外，还可以直接恢复数据库到日志末尾，使用如下命令：

```
C:\>db2 recover database sample to end of logs
前滚状态
输入数据库别名 = sample
节点数已返回状态 = 1
节点号 = 0
前滚状态 = 未暂挂
下一个要读取的日志文件 =
已处理的日志文件 = S0000000.LOG - S0000001.LOG
上次落实的事务 = 2008-10-25-22.30.00.000000
DB20000I RECOVER DATABASE 命令成功完成
```

不管出于何种原因，如果恢复操作还没有成功完成就被中断，那么可以通过运行相同的命令来重新启动这个操作。如果是在前滚阶段被中断，那么恢复实用程序将尝试继续之前的前滚操作，而不会重复恢复阶段。如果要强制恢复实用程序重复恢复阶段，可以带

RESTART 选项发出 RECOVER DATABASE 命令，迫使恢复实用程序忽略之前没能完成的恢复操作。如果恢复实用程序在恢复阶段被中断，那么它将从头开始。

恢复实用程序不支持以下 RESTORE DATABASE 命令选项：

- TABLESPACE tablespace-name ——表空间恢复操作不受支持
- INCREMENTAL——增量恢复操作不受支持
- OPEN num-sessions SESSIONS——不能用 TSM 或另一种供应商产品指定 I/O 会话号
- BUFFER buffer-size—— 不能设置用于恢复操作的缓冲区的大小
- DLREPORT filename——不能指定已断开链接的报告文件的文件名
- PARALLELISM n——不能指定恢复操作的并行度
- WITHOUT PROMPTING——不能规定恢复操作在没有提示的情况下运行

7.7 恢复历史文件

我们在上面的恢复过程中提到了恢复历史记录文件(Recover History File，在下面简称为历史文件)，历史文件是与每个数据库一起创建的(图 7-21 所示)，并且在发生下列情况时自动更新：

- 备份、复原、恢复和前滚了数据库或表空间
- 自动重建了数据库，并且复原了多个映像
- 创建、改变、停顿、重命名或删除了表空间
- 装入(LOAD)、重组、runstats 表
- DROP 表(启用了 DROPPED TABLE RECOVERY)
- 调用按需(archive)进行的日志归档
- 写入新日志文件(使用可恢复日志记录时)
- 归档日志文件(使用可恢复日志记录时)

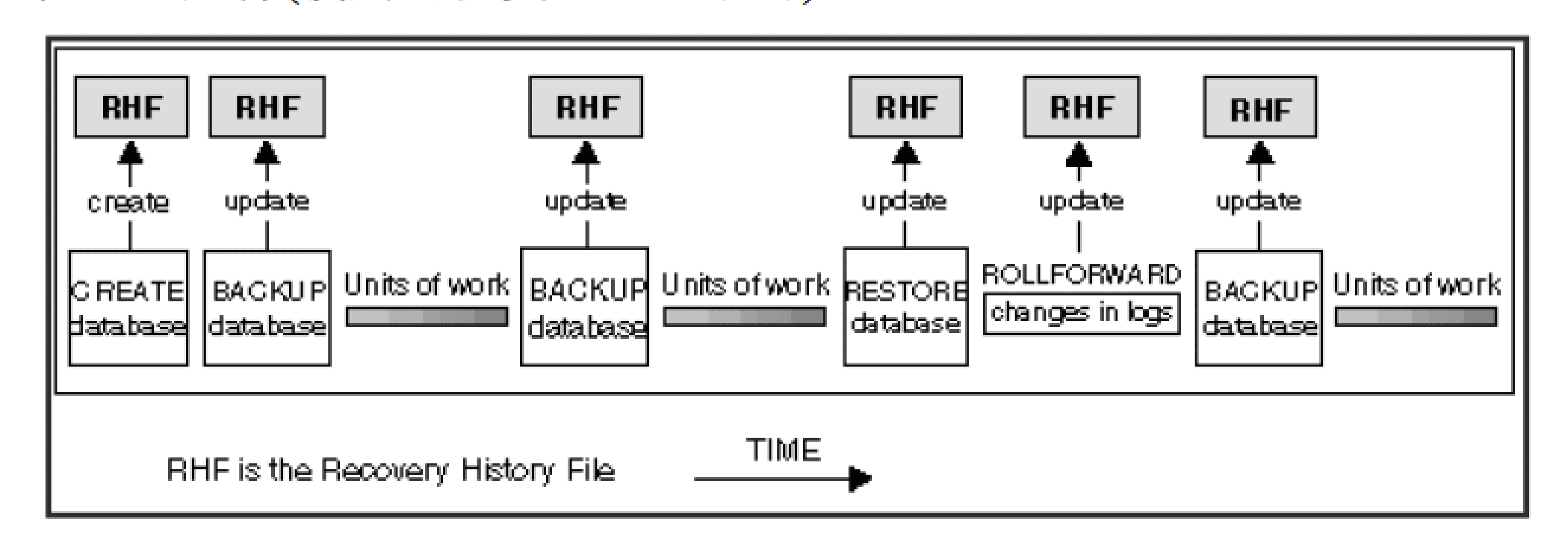


图 7-21 恢复历史记录文件

可以使用历史文件中汇总的备份信息，将数据库的全部或部分恢复至给定的时间点。历史文件中的信息包括：

- 唯一地标识每个条目的 ID 字段
- 已复制的部分数据库和复制方法
- 建立副本的时间
- 副本的位置(指示设备信息和访问副本的逻辑方式)
- 上次进行复原操作的时间
- 重命名表空间的时间，显示表空间的先前名称和当前名称
- 备份操作的状态：活动、不活动、到期的或删除的
- 数据库备份保存的或前滚恢复操作期间处理的最后一个日志序号

要查看历史文件中的条目，可使用 LIST HISTORY 命令。要列出对数据库 SAMPLE 完成的所有备份(BACKUP)、恢复(RESTORE)或装入(LOAD)操作，可以发出以下命令：

```
DB2 LIST HISTORY ALL FOR SAMPLE
```

每当执行备份、恢复或 LOAD 操作时，就会自动地更新历史文件的相应内容。用户不能把记录直接写入历史文件中。

每个备份操作(数据库备份、表空间备份或增量备份)都会把历史文件同时备份。历史文件与数据库的备份映像相关联。删除数据库会删除历史文件。将数据库恢复至新位置会复原历史文件。恢复不会覆盖现有历史文件，除非磁盘上的文件没有任何条目(在这种情况下，将根据备份映像复原数据库历史记录)。

如果由于文件系统、恢复历史文件被破坏或出现 I/O 错误，使得不能向恢复历史文件插入或更新信息，那么用户将会收到警告信息。

用户可以用 PRUNE HISTORY 命令管理这一恢复历史文件。该命令的语法如下：

```
PRUNE HISTORY timestamp [WITH FORCE OPTIONS]
```

时间戳等于或小于所提供时间印戳值的所有项都要从恢复历史文件中删除。WITH FORCE OPTIONS 规定：根据指定的时间戳修改历史文件，即使最近恢复集合中的一些项已经从历史文件中删除。

要使用 PRUNE HISTORY 命令，用户必须拥有 SYSADM、SYSCTRL、SYSMAINT 或 DBADM 权限。每个数据库都有恢复历史数据。历史文件的大小取决于 PRUNE HISTORY 这一数据库配置参数和备份、恢复以及表 LOAD 操作的频率。

在 DB2 V10.1 版本中，PRUNE HISTORY 命令得到了增强，除了可以用于清除历史文件中的记录之外，还可以用于清除日志文件，用于替代 PRUNE LOGFILE 命令。

数据库配置参数 REC_ HIS_ RETENTN 用来设置历史文件的保留期间(默认值 366 天)。

如果用户想要无限期地保持恢复历史文件，可以把这一参数设置为－1。按默认方式，在记录完整个数据库备份之后，恢复历史文件会被自动根据配置参数 REC_ HIS_ RETENTN 设置成自动清理。

恢复历史文件的内容如表 7-4 中所示。

表 7-4　恢复历史文件的内容

列　　名	描　　述
OPERATION	执行的操作类型：B = Backup(备份)，R = Restore(恢复)，L = Load(装入)，A = Create Tablespace(创建表空间)，N=Rename(重命名)等
OBJECT	对象标识，标识操作的对象
时间戳+序列	记录操作的时间戳和序列
OPTYPE	操作类型，例如 F=Full 数据库离线备份；N=Online，R=Replace；P = Table Space(表空间)　，T= Table(表)
DEVICE_TYPE	设备类型：D=Disk(磁盘)，K=Diskette(软盘)，T=Tape(磁带)，A= ADSM，U= UserExit(用户出口)，O =其他供应商设备
FIRST_LOG	**最早日志：对于在线备份来说，该日志表示是在线备份期间开始写入的第一个日志**
LAST_LOG	**最后日志：对于在线备份来说，表示在线备份完成后写入的最后一个日志。对于离线备份来说，最早日志和最晚日志永远相等**
BACKUP_ID	备份。前 14 个字母 = yyymmddhhnnss，后 3 个字符 = 顺序号
COMMENT	操作注释，例如备份类型
STARTIME	操作开始时间
STOPTIME	操作结束时间
LOCATION	存放位置

在恢复历史文件的输出中，最早日志(FIRST_LOG)和最后日志(LAST_LOG)非常重要，它们对维护离线数据库和在线数据库的完整性非常重要。最早日志：对于在线备份来说，最早日志是在线备份期间开始写入的第一个日志。最后日志：对于在线备份来说，是在线备份完成后写入的最后一个日志。对于离线备份来说，最早日志和最晚日志永远相等。对于离线备份来说，我们必须保存好最晚日志之后的日志文件以执行前滚恢复。对于在线备份来说，我们必须维护好最早日志和最晚日志之间的日志文件以保证在线备份的有效性。如果最早日志和最晚日志之间的日志文件丢失，那么这个在线备份是无效的。这一定要注意。当然，如果在 BACKUP 命令中使用 INCLUDE LOGS 选项，一般就不需要担心这些日志了，这些日志会自动保存到备份映像中。

如果需要已经被删除数据库的历史文件，那么 RESTORE DATABASE 命令有一个选项允许只恢复相应的历史文件。这样就可以通过查看历史记录文件的内容，提供有关要用于复原数据库的备份的信息。假如历史文件被破坏，可以通过下面的命令来恢复某个特定备份版本的历史文件：

```
DB2 restore db sample history file  --这个操作只恢复历史文件
```

7.8　数据库重建

7.8.1　数据库重建的概念

数据库重建(REBUILD)功能是由恢复(RECOVER)实用程序提供的，允许使用一组备份映像重建全新的数据库。可以选择重建整个数据库，或者重建只包含原来数据库中部分表空间的数据库。数据库重建过程取决于数据库是可恢复的(采用了归档日志模式)还是不可恢复的(采用了循环日志模式)。在后面的内容中我们将先后谈到这两种场景。

7.8.2　使用表空间备份重建可恢复数据库

对于可恢复数据库，REBUILD 实用程序允许只使用表空间备份重建整个数据库。这里不需要完整数据库备份。完整数据库备份可能需要更大的维护窗口，对于可用性要求比较高的环境，这会增加实施的难度。使用表空间备份重建数据库的能力对于可用性和可恢复性来说是个很好的增强。

假设拥有名为 TEST 的可恢复数据库。某天夜里，出现了一次停电事故。数据库所在的磁盘遭到了损坏。数据库再也不能访问了，于是想恢复数据库。TEST 数据库有以下表空间：

- SYSCATSPACE(系统编目)
- USERSPACE1(用户数据表空间)
- USERSPACE2(用户数据表空间)
- USERSPACE3(用户数据表空间)

可以使用的还有：

- 所有日志文件。由于日志与数据库存储在不同的磁盘上，因此它们能够幸免于难。
- 你没有任何数据库级的备份，但是有以下表空间备份：

```
TEST.3.DB2.NODE0000.CATN0000.20080515135047.001 -- SYSCATSPACE 和
                                                    USERSPACE1 的备份
TEST.3.DB2.NODE0000.CATN0000.20080516135136.001 -- USERSPACE2 和
```

```
                                                    USERSPACE3 的备份
    TEST.3.DB2.NODE0000.CATN0000.20080517135208.001 -- USERSPACE3 的备份
```

如果使用恢复和前滚方法(前面有过讲解)，将数据库恢复到最近的某个时间点，那么需要恢复数据库备份，然后将数据库前滚到日志的最后。遗憾的是，在这种情况下，这是不可能实现的，因为我们没有数据库备份，我们只有表空间备份。如果在任何表空间备份上运行常见的 RESTORE 命令，将得到以下错误：

```
    db2 restore db test taken at 20080517135208
    SQL2560N The target database is not identical to the source database for
a restore from a table space level backup.
```

有了数据库重建功能，现在可以只用表空间备份和日志重建 TEST 数据库。要重建数据库，可以在 RESTORE DATABASE 命令中指定 REBUILD 选项。

下面的步骤将 TEST 数据库重建到最近的时间点。

(1) 带 REBUILD 选项发出 RESTORE DATABASE 命令：

```
    restore db test rebuild with all tablespaces in database taken at 20080517135208
```

重建过程中的第一步是识别重建目标映像。重建目标映像应该是要在重建操作中使用的最近的备份映像。之所以称为目标映像，是因为其中定义了要重建的数据库的结构，包括可以恢复的表空间、数据库配置和日志序号，可以是任何类型的备份(完整备份、表空间备份、增量备份、在线或离线备份)。在这个例子中，最近的备份映像是 TEST.3.DB2.NODE0000.CATN0000.20080517135208.001，因此将之作为重建操作的目标映像。

在这个命令成功执行之后，TEST 数据库的结构被恢复。我们可以得到数据库配置和历史文件之类的信息。如果发出 LIST HISTORY 命令(例如 db2 list history all for test)，我们将得到以下输出：

```
    Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
    -- --- ------------------ ---- --- ------------ ------------ --------------
     R  D  20080519121107001    F                              20080517135208
    ---------------------------------------------------------------------------
     Contains 1 tablespace(s):
     00001 USERSPACE3
    ---------------------------------------------------------------------------
       Comment: RESTORE TEST WITH RF
    Start Time: 20080519121107
      End Time: 20080519121108
        Status: A
    ---------------------------------------------------------------------------
     EID: 7 Location:
```

```
Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
-- --- ------------------ ---- --- ------------ ------------ --------------
 R  P  20080519121108001   F                              20080515135047
 ----------------------------------------------------------------------------
 Contains 2 tablespace(s):
 00001 USERSPACE1
 00002 SYSCATSPACE
 ----------------------------------------------------------------------------
   Comment: RESTORE TEST WITH RF
Start Time: 20080519121108
  End Time: 20080519121113
    Status: A
 ----------------------------------------------------------------------------
 EID: 8 Location:
Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
-- --- ------------------ ---- --- ------------ ------------ --------------
 R  P  20080519121113001   F                              20080516135136
 ----------------------------------------------------------------------------
 Contains 1 tablespace(s):
 00001 USERSPACE2
 ----------------------------------------------------------------------------
   Comment: RESTORE TEST WITH RF
Start Time: 20080519121113
  End Time: 20080519121114
    Status: A
 ----------------------------------------------------------------------------
 EID: 9 Location:
Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
-- --- ------------------ ---- --- ------------ ------------ --------------
 R  D  20080519121107      R       S0000001.LOG S0000003.LOG 20080518135208
 ----------------------------------------------------------------------------
 Contains 4 tablespace(s):
 00001 USERSPACE3
 00002 USERSPACE2
 00003 USERSPACE1
 00004 SYSCATSPACE
 ----------------------------------------------------------------------------
   Comment: REBUILD TEST WITH RF
Start Time: 20080519121107
  End Time: 20080519121115
    Status: A
 ----------------------------------------------------------------------------
 EID: 10 Location:
```

上面显示的 LIST HISTORY 命令输出中有 4 个条目，它们都与重建操作有关。

第一个条目是 EID:7，它表明是备份映像 20080517135208 上的一个恢复操作，被恢复的表空间为 USERSPACE3(这个备份映像只包含 USERSPACE3)。但是，我们需要使用 ALL TABLESPACES 选项恢复所有表空间，所以数据库中剩下的表空间也要恢复。这一点反映在 LIST HISTORY 输出中剩下的部分。

通过使用备份恢复文件中的信息，恢复实用程序发现要恢复的所有表空间的备份映像并恢复它们。在恢复之后，表空间被置于 rollforward pending 状态。可以看看 LIST HISTORY 输出中的注释行，每个表空间都标上了'WITH RF'，表明在恢复之后要执行前滚。

要进行恢复，所有备份映像都必须已经存在，并在历史文件中能够找到相应的记录。否则就会返回错误，说明恢复实用程序不能发现所需的映像。

(2) 带 TO END OF LOGS 选项发出 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test to end of logs
```

在恢复所有表空间之后，这些表空间被置于 rollforward pending 状态。我们需要前滚数据库，使数据库回到正常状态。

要在重建操作期间前滚数据库，最早备份映像与最近备份映像之间的所有日志文件都必须能够为前滚实用程序所用。如果想前滚到比最近备份更近的时间点上，那么这个备份之后的所有日志文件也必须可用。

在我们的例子中，所有日志仍然完好无损，它们仍然存在于 LOGPATH 数据库配置参数指定的日志路径中。前滚实用程序将在那里找到它们。正因为如此，我们不需要在 ROLLFORWARD 命令中指定日志文件的路径。如果日志文件被存储在其他地方，就必须使用 ROLLFORWARD 命令中的 OVERFLOW LOG PATH 选项指定日志文件的位置。

(3) 带 STOP 选项发出 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

至此，TEST 数据库已经可以连接，并且所有表空间都处于 NORMAL 状态。

7.8.3 只使用部分表空间备份重建可恢复数据库

正如 7.8.2 节中演示的那样，数据库重建功能让我们可以只使用表空间备份和日志来重建完整的数据库。这个实用程序是如此健壮，以至于我们不需要所有的表空间备份就能重建数据库。只用一部分表空间就能重建数据库。

再次使用 7.8.2 节的例子，假设 USERSPACE1 和 USERSPACE2 中的数据对于用户来说非常重要。在停电事故发生后，我们必须尽快恢复这两个表空间。USERSPACE3 则不太重要，而且它很大。如果恢复所有表空间，那么需要花很长的时间。如果可以只用其中的

USERSPACE1 和 USERSPACE2 重建可连接的数据库，那就很好了，这样用户很快就可以使用数据库。如果时间允许的话，可以再来恢复 USERSPACE3。下面的步骤展示了如何使用数据库重建实用程序来完成这一任务。

(1) 带 REBUILD 选项发出 RESTORE DATABASE 命令，指定只恢复部分表空间：

```
db2 restore db test rebuild with tablespace
(SYSCATSPACE,USERSPACE1,USERSPACE2) taken at 20080516135136
```

虽然我们只想恢复 USERSPACE1 和 USERSPACE2，但是还必须恢复 SYSCATSPACE，因为这个表空间中存放着所有系统信息。没有它，DB2 就不知道关于这个数据库的结构的任何信息。

上面指定的目标映像是包含 USERSPACE2 和 USERSPACE3 的映像，这是包含我们要恢复的表空间的最近备份。虽然 20080517135208 是三个备份当中最近的备份，但是我们不能使用它，因为它不包含 USERSPACE1、USERSPACE2 或 SYSCATSPACE。

下面的命令效果是一样的：

```
db2 restore db test rebuild with all tablespaces in database except tablespace
   (USERSPACE3) taken at 20080516135136
```

(2) 带 TO END OF LOGS 选项发出 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test to end of logs
```

(3) 带 STOP 选项发出 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

可以选择前滚到另一个时间点，而不是前滚到日志的最后。你所选择的时间点必须大于恢复中使用的备份映像的时间戳。

至此，TEST 数据库已经可以连接，并且除了 USERSPACE3 之外的所有表空间都处于 NORMAL 状态。USERSPACE3 将处于 RESTORE PENDING 状态。

可以在以后通过常规的表空间恢复(不带 REBUILD 选项)来恢复 USERSPACE3。

(4) 发出 RESTORE DATABASE 命令并指定要恢复的表空间：

```
DB2 restore db test tablespace (USERSPACE3) taken at 20080517135208
```

(5) 带 TO END OF LOGS 选项发出 ROLLFORWARD DATABASE 命令并指定要前滚的表空间：

```
db2 rollforward db test to end of logs tablespace (USERSPACE3)
```

(6) 带 STOP 选项发出 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

现在，TEST 数据库的所有 4 个表空间都处于 NORMAL 状态。

在生产环境中，或者在像前面那样的恢复场景中，只让一部分表空间快速恢复以对外提供可用性是很有用的。在测试环境中这样做也有用处，可以只恢复感兴趣的那部分表空间。

7.8.4 使用包含日志文件的在线备份重建数据库

当重建可恢复数据库时，既可以使用数据库备份，也可以使用表空间备份。备份可以是在线的，也可以是离线的。

如果拥有包含日志文件的在线备份映像，并且想使用这些日志来前滚数据库，那么可以使用 RESTORE DATABASE 命令的 LOGTARGET 选项从映像中获取日志。

再次使用 TEST 数据库作为例子，假设备份映像 TEST.3.DB2.NODE0000.CATN0000.20080517135208.001 是包含日志的在线备份映像。要使用表空间备份和存储在备份映像中的日志恢复整个数据库，可以：

(1) 带 LOGTARGET 选项发出 RESTORE DATABASE 命令。在恢复期间，这些日志被提取到 LOGTARGET 指定的位置：

```
    db2 restore db test rebuild with all tablespaces in database taken at
20080517135208   logtarget /logs
```

(2) 带 TO END OF LOGS 选项发出 ROLLFORWARD DATABASE 命令，并指定日志的位置：

```
db2 rollforward db test to end of logs overflow log path (/logs)
```

注意：

OVERFLOW LOG PATH 选项用于指定日志的位置。

(3) 带 STOP 选项发出 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

7.8.5 使用增量备份映像重建可恢复数据库

增量备份映像也可以用于重建数据库。当重建过程中涉及增量映像时，默认情况下恢复实用程序会尝试为所有增量映像使用自动增量恢复。如果没有使用 RESTORE DATABASE

命令的 INCREMENTAL AUTOMATIC 选项，但目标映像是增量备份映像，那么恢复实用程序将使用自动增量恢复发出重建操作。

如果目标映像不是增量映像，但另一个需要的映像是增量映像，那么恢复实用程序也将确保使用自动增量恢复来恢复那些增量映像。不管是否指定 INCREMENTAL AUTOMATIC 选项，恢复实用程序的行为都是一样的。

如果指定 INCREMENTAL 选项而没有指定 AUTOMATIC 选项，那么需要手动执行整个重建过程。恢复实用程序只是从目标映像中恢复初始的元数据，就像在常规的手动增量恢复中一样。然后还需要使用所需的增量恢复链(见 7.4.3 节内容)来完成目标映像的恢复。最后，为了重建数据库，还需要恢复剩下的映像。这个过程可能比较冗长。

建议使用自动增量恢复来重建数据库。只有在恢复失败的情况下，才应该尝试通过手动方式重建数据库。

7.8.6　使用重定向选项重建可恢复数据库

由于重建功能是恢复实用程序的一部分，因此可以使用重定向方法重建数据库，就像在重定向恢复中那样。下面使用 REDIRECT 选项将整个 TEST 数据库重建到最近的时间点：

(1) 带 REBUILD 和 REDIRECT 选项发出 RESTORE DATABASE 命令：

```
    db2 restore db test rebuild with all tablespaces in database taken at
20080517135208    redirect
```

(2) 对要为之重新定义容器的每个表空间发出 SET TABLESPACE CONTAINERS 命令。例如：

```
    db2 set tablespace containers for 3 using (file '/newuserspace2' 10000)
    db2 set tablespace containers for 4 using (file '/newuserspace3' 15000)
```

(3) 带 CONTINUE 选项发出 RESTORE DATABASE 命令：

```
    db2 restore db test continue
```

(4) 带 TO END OF LOGS 选项发出 ROLLFORWARD DATABASE 命令(假设日志路径目录中的所有日志都是可以访问的；否则，需要使用 OVERFLOW LOG PATH 选项来指定备选日志路径)：

```
    db2 rollforward db test to end of logs
```

(5) 带 STOP 选项发出 ROLLFORWARD DATABASE 命令：

```
    db2 rollforward db test stop
```

至此，这个数据库已经可以连接，并且所有表空间都处于 NORMAL 状态。

7.8.7 重建不可恢复数据库

到目前为止，我们讨论的所有重建方法都同样适用于不可恢复(采用了循环日志模式)数据库。唯一的区别是：

- 如果数据库是不可恢复的，那么在重建操作中只能使用数据库备份作为恢复的目标映像，因为不可恢复数据库没有可用的表空间备份。
- 当恢复完成时，马上就可以连接到数据库——不需要前滚操作。但是，任何尚未恢复的表空间都被置于 drop pending 状态，并且它们再也不能被恢复。

让我们看一个例子。假设存在不可恢复的数据库 MYDB。MYDB 有 3 个表空间：SYSCATSPACE、USERSP1 和 USERSP2。在 20080521130000 做了一次完整数据库备份。

只使用 SYSCATSPACE 和 USERSP1 重建数据库：

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1) taken at
20080521130000
```

在恢复之后，马上就可以连接数据库了。如果发出 LIST TABLESPACES 命令，你将看到 SYSCATSPACE 和 USERSP1 处于 NORMAL 状态，而 USERSP2 则处于 DROP PENDING 状态。现在，可以使用处于 NORMAL 状态的那两个表空间。

在这种情况下，如果要做数据库备份，那么首先必须使用 DROP TABLESPACE 命令删除 USERSP2，否则备份会失败。

7.8.8 数据库重建的限制

重建数据库的能力使恢复实用程序更加强大。但是，这方面也有一些限制：

- 重建的表空间中必须包括 SYSCATSPACE。
- 不能使用控制中心的图形化工具执行重建操作，只能使用命令行处理器(CLP)发出命令。
- 对于 DB2 V9.1 版本之前的目标映像，除非这个映像是离线数据库备份，否则不能使用 REBUILD 选项。如果目标映像是离线数据库备份，那么只有这个映像中的表空间可以用于重建。在重建操作成功完成之后，需要迁移数据库。如果使用 DB2 V9.1 版本之前的其他类型的目标映像进行重建，将导致错误。
- 除非目标映像是完整数据库备份，否则，如果目标映像与被恢复数据库处在不同的操作系统中，那么不能对目标映像发出 REBUILD 选项。如果目标映像是完整数据库备份，那么只有这个映像中的表空间可用于重建。

7.9　监控备份、复原和恢复进度

可以使用 LIST UTILITIES 或 db2pd -util 命令来监控数据库上的备份、复原和前滚操作。

要对备份、复原和恢复操作使用进度监控，发出 LIST UTILITIES 命令并指定 SHOW DETAIL 选项：

```
LIST UTILITIES SHOW DETAIL
```

以下是用于监控脱机数据库备份操作性能的输出的示例：

```
LIST UTILITIES SHOW DETAIL
      标识                          = 2
      类型                          =备份
     数据库名称                     = 样本
      描述                          = 脱机数据库
      开始时间                      = 10/30/2008 12:55:31.786115
      正在调速:
      优先级                        = 最低
      进度监控:
      估计完成百分比                = 41
                  全部工作单元           = 20232453 个字节
                  已完成的工作单元       = 230637 个字节
                  开始时间             = 10/30/2008 12:55:31.786115
```

对于备份操作，将指定要处理的最初估计字节数。随备份操作进度更新要处理的字节数。显示的字节与映像的大小不相等，不应该用作备份映像大小的估计值。具体情况需要视映像是增量备份还是压缩备份而定，实际映像可能小很多。

对于复原操作，将不给定任何最初的估计值，而是指定 UNKNOWN。因为从映像中读取每个缓冲区，所以将会更新实际读取的字节量。对于其中可能复原多个映像的自动增量复原操作，将使用阶段来跟踪进度。每个阶段提供一个从增量链中复原的映像。最初，只显示一个阶段。复原第一个映像之后，将显示阶段的总数。在复原每个映像时，将根据已处理的字节数来更新完成的阶段数。

对于崩溃恢复和前滚恢复，将有两个进度监控阶段：FORWARD 和 BACKWARD。在 FORWARD 阶段，读取日志文件并将日志记录应用于数据库。对于崩溃恢复，通过使用起始日志序号至最后一个日志文件的结尾来估计工作总量。对于前滚恢复，当此阶段开始时，将工作总量的估计值指定为 UNKNOWN。按字节计的已处理工作量将随进程的继续而更新。

在 BACKWARD 阶段，回滚 FORWARD 阶段任何未落实的更改，将提供按字节计的

需处理日志数据量的估计值。按字节计的已处理工作量将随进程的继续而更新。

db2pd -util 命令的输出内容与上述内容类似，只是表现形式有所区别，示例如下：

```
    db2pd -util

    Database Member 0 -- Active -- Up 17 days 21:13:40 -- Date 08/20/2012 15:08:00

    Utilities:
    Address            ID        Type                 State     Invoker
Priority  StartTime          DBName   NumPhases  CurPhase   Description
    0x078000003C5FF960 5745      BACKUP                0         0         0
Mon Aug 20 15:07:31 PNB      1          1          online db

    Progress:
    Address            ID        PhaseNum  CompletedWork
TotalWork                StartTime          Description
    0x078000003C5DF848 5745      1        41864300196 bytes
555781079740 bytes          Mon Aug 20 15:07:31 n/a
```

7.10　备份、恢复和复原期间的表空间状态

表空间的当前情况由其状态反映。与备份恢复相关的最常见表空间状态是：

- 备份暂挂(Backup Pending)。在前滚操作的某个时间点后，或不带有复制选项的装入操作后，表空间将置于此状态。在可以使用表空间之前必须对其备份(如果未进行备份，就不能更新表空间，但允许只读操作)。
- 复原暂挂(Restore Pending)。如果取消对表空间的前滚操作，或对表空间的前滚操作遇到了不可恢复错误(此时必须再次复原并前滚表空间)，就会将表空间置于此状态。在复原操作期间，如果无法复原表空间，表空间也会处于此状态。
- 正在前滚(RollForward in Progress)。表空间正在进行前滚时，被置于此状态。一旦前滚操作成功完成，表空间就不再处于"正在前滚"状态。如果取消对表空间的前滚操作，表空间也会结束此状态。
- 前滚暂挂(RollForward Pending)。表空间在复原后发生了输入/输出(I/O)错误后，表空间会被置于此状态。复原后，表空间可前滚到日志末尾的某个时间点。发生了 I/O 错误后，表空间必须前滚到日志的末尾。

7.11　优化备份、复原和恢复性能

在 DB2 V8 以后，在执行备份、恢复和复原操作时，DB2 将自动为缓冲区个数、缓冲

区大小和并行性设置选择最佳值。这些值根据可用实用程序的堆内存数量、可用 CPU 数量和数据库配置而定。因此，应根据系统中可用的内存大小，考虑通过增大 UTIL_HEAP_SZ 配置参数来分配更多内存。目的是在最大程度上减小备份、恢复和复原操作所需的时间。除非显式地输入以下命令参数的值，否则 DB2 将为它们选择值：

- WITH num-buffers BUFFERS
- PARALLELISM n
- BUFFER buffer-size

如果未指定缓冲区个数和缓冲区大小而导致 DB2 自动设置这些值，那么对大型数据库的影响应该最低。但是，对于小型数据库来说，会导致备份映像大幅增大。即使写入磁盘的最后一个数据缓冲区只包含很少数据，也会将整个缓冲区写入映像。在小型数据库中，这表示相当一部分的映像可能为空。

还可以选择执行以下任何操作来缩短完成一次备份、恢复和复原操作所需的时间：

- 增大 PARALLELISM 参数的值，通常会影响可以并行备份的表空间个数。PARALLELISM 参数定义在备份操作期间从数据库读取数据和压缩数据时，已启动的线程数，启动多个线程可以并行地执行备份数据操作以提高性能。但是，备份执行过程中通常是以表空间为工作的最小单位，所以为 PARALLELISM 参数指定的值大于要备份的表空间个数并无益处。但是应注意，每个进程或线程都需要内存和 CPU 开销。如果 CPU 并未成为瓶颈，那么可以增加这个参数的设置。此值如果不在 BACKUP 命令中指定，DB2 会自动分配值。目前来看，DB2 自动分配的值在大多数情况下是适当的。
- 增加备份、恢复缓冲区大小。理想的备份、恢复缓冲区大小是表空间扩展数据块大小的倍数加一页。如果有多个扩展数据块大小不同的表空间，那么将值指定为扩展数据块大小的公倍数加一页。例如：

```
SYSCATSPACE          扩展数据块大小(页)          = 4
TEMPSPACE1           扩展数据块大小(页)          = 32
USERSPACE1           扩展数据块大小(页)          = 32
IBMDB2SAMPLEREL      扩展数据块大小(页)          = 32
DATA_SPACE           扩展数据块大小(页)          = 8
INDEX_SPACE          扩展数据块大小(页)          = 16
TS2                  扩展数据块大小(页)          = 32
```

在上面的例子中，我们有 7 个表空间，它们的扩展数据块大小不一样。在这种情况下，如果要在备份、恢复和复原时设置备份、恢复缓冲区大小，应该设置成它们扩展数据块的公倍数。

在上面的例子中，32 是个公倍数。考虑设置备份、恢复缓冲区大小为 32+1=33 页。

- 增加缓冲区的数量。使用的缓冲区至少是备份目标(或会话)的两倍，从而确保备份目标设备无须等待数据。
- 使用多个目标设备备份恢复。虽然在设置完上述参数后可以提高备份、恢复和复原的性能，但是上述设置中的缓冲大小受到数据库参数 UTIL_HEAP_SZ 值的限制。所以在备份、恢复和复原期间应监控 UTIL_HEAP_SZ 内存的使用情况。请看下面的例子：

```
db2 get snapshot for database on sample
..............................略...................................................
数据库的内存使用情况：
  节点号                                      = 0
       内存池类型                             = 备份/复原/实用程序堆
       当前大小(以字节计)                     = 17563648
       高水位标记(以字节计)                   = 17563648
       已配置的大小(以字节计)                 = 20512768
..............................略...................................................
```

在备份、恢复和复原期间，监控数据库备份/复原/实用程序堆的当前大小和高水位标记。如果当前大小和高水位标记接近“已配置的大小”，并且内存资源充足，可以考虑增大 UITL_HEAP_SZ 参数。而且 UITL_HEAP_SZ 参数指定的内存区域并非按照预分配的方式使用，而是在需要的时候才会申请分配。

7.12 备份恢复最佳实践

备份恢复策略其实就是在备份时间和恢复时间之间做选择折中和平衡。要保证数据的安全，可以使用以下最佳实践：

- 保证数据库处于归档日志模式，这样当数据库发生故障时便可以将之恢复到特定的时间点。
- 定期执行完整和递增的数据库备份。业务需求将最终决定时间表和频率。
- 如果数据库特别重要，考虑使用映像日志。但是映像日志会带来部分性能开销。
- 根据业务需求和数据库大小指定合适的备份恢复策略。

在用户的备份恢复策略中要考虑以下事项：

- 考虑要使用的日志类型，即循环日志和归档日志。
- 决定对备份恢复操作采用的访问(读取)类型。
- 要意识到恢复动作可能包括前滚操作。

- 要确定必须进行前滚操作的时间点，比如最小恢复时间(Minimum Recovery Time)点的要求。
- 对于比较重要的交易型数据库，在本地准备副本或备用数据库以供随时使用。
- 对于增量备份，确保各个备份文件的完备性。
- 尽量多保留备份文件。一份放在本地以备数据库随时使用，一份放到磁带或带库上，另一份远程异地存放。
- 将活动日志文件和归档日志文件保存在不同的位置，并且各位置拥有充足的磁盘空间。

表 7-5 总结归纳了在表空间的整个数据库级别上进行备份与恢复操作的各种考虑。

表 7-5　备份操作的限制与约束条件

	数据库 离线备份		数据库 在线备份	增量备份	表空间 离线备份	表空间 在线备份
日志类型	归档	循环	归档	归档	归档	归档
备份期间读取	不适用	不适用	可以存取	可以读取	不允许访问数据库	对数据库的充分存取
恢复之后的数据库状态	数据库处于前滚挂起状态	数据库处于一致状态	数据库处于前滚挂起状态	数据库处于前滚挂起状态	恢复的表空间处于前滚挂起状态，其他表空间正常	数据库处于前滚挂起状态
前滚操作	在任何时刻	不适用	在备份结束后的任何时间点	在备份结束后的任何时间点	在备份结束后的任何时间点	在备份结束后的任何时间点



SQL 基础知识

SQL(Structured Query Language)是关系数据库的核心语言，可以用来定义和操作数据库，是用户与数据库交互联系的重要桥梁。SQL 语言不同于 C、Java 等过程化语言，只定义必要的输入和输出，执行语句的方式由数据库引擎的组件(优化器)处理。从关系数据库诞生初期，美国国家标准组织(ANSI)就指定了一套 SQL 标准，之后又不断地改进，而 DB2 作为市场分布非常广泛的一种关系数据库(SQL 语言都是以 SQL 标准为基础)，同时添加了少量的专属 DB2 的 SQL 方言。

在本章中我们主要简单介绍 SQL 语言的基础知识，如果您已熟练掌握 SQL 语言，请跳过本章的前 4 节。

8.1　简单查询入门

查询(SELECT)语句是 SQL 语言的核心部分，所提供的选项既复杂又强大。本章我们将从表查询语句开始，并讨论每个组成部分是如何在数据库中工作的。

本章中要查询的表和数据主要来源于 DB2 自带的 sample 数据库，我们可以使用实例用户创建 sample 数据库：

```
db2sampl -dbpath /db2/sample -name SAMPLE -sql
```

SELECT 查询语句的基本语法：

```
>>-SELECT------+--选择列-----+----------------------------------------->
>--FROM--------+--表名-------+----------------------------------------->
>--WHERE-------+--搜索条件----+--------------------------------------->
>--GROUP BY----+--选择列------+--------------------------------------->
>--HAVING------+--搜索条件----+--------------------------------------->
>--ORDER BY----+--选择列------+--------------------------------------><
```

8.1.1 SELECT 和 FROM

查询语句中的 SELECT 子句是最开始的部分，指定要显示的属性列；FROM 子句定义了查询中使用的表或视图，以及连接这些表的方式。

最简单的查询语句就是从数据库的一张表中查出列的数据：

```
SELECT EMPNO, FIRSTNME
  FROM EMPLOYEE
```

在这个最简单的查询中，包含了 SELECT 和 FROM 子句两个必选项。

如果我们想查出所有列的数据，可以用星号(*)代替所有列：

```
SELECT *
  FROM EMPLOYEE
```

8.1.2 WHERE

WHERE 子句用于在结果集中过滤掉不需要的行，主要指定搜索条件。一般情况下，需要查询表中的所有记录意义不大，尤其是在记录很多的表中，我们希望准确定位到想查看的信息。这时 WHERE 子句后面的搜索条件就可以帮助我们从所有记录中筛选结果。

比如查找员工 JONH 的信息：

```
SELECT *
  FROM  EMPLOYEE
  WHERE FIRSTNME = 'JOHN'
```

搜索条件主要靠 WHERE 谓词来实现，在 8.2 节中我们将讲解几种常用 WHERE 谓词的用法。

8.1.3 ORDER BY

ORDER BY 子句用于对查询结果中的原始列数据或是根据列数据计算的表达式结果进行排序，一般是对查询结果按照指定列升序或降序排列。

使用 ORDER BY 子句，可以按照一个或多个属性列排序，主要有升序(ASC)和降序(DESC)两种方式，默认为升序，语法如下：

```
>>-ORDER BY--+-字段名-------+-------------+---------------------------->
                            +--ASC--------+
                            '--DESC-------'
```

比如在查询员工信息时按照员工的级别和员工号降序排列：

```
SELECT EMPNO, FIRSTNME, EDLEVEL
FROM   EMPLOYEE
ORDER BY EDLEVEL DESC, EMPNO DESC
```

在这条查询中，EDLEVEL 是第一排序健，EMPNO 是较次要的排序健。当主要健的数据相同时，就按照次要排序健排序。

在查询到的部分结果中，我们就可以看到：都是 18 级的员工 REBA 就排在 KIM 和 DIAN 之前。

```
000030SALLY 20
000110VINCENZO 19
200220REBA 18
200140KIM 18
200010DIAN 18
......
200280EILEEN 17
200240ROBERT 17
```

8.1.4　GROUP BY 和 HAVING

GROUP BY 子句对查询结果按指定列的值分组，该属性列值相等的组为一组。而 HAVING 子句则用来筛选出只满足指定条件的组。通常情况下，GROUP BY 和 HAVING 子句是一起使用的，如果 HAVING 子句在没有 GROUP BY 子句的情况下使用，数据库会把查询结果默认视为一组。

列出所有部门的平均和最高薪水：

```
SELECT AVG(SALARY), MAX(SALARY)
  FROM EMPLOYEE
```

如果我们需要列出每个部门的平均和最高薪水，就可以使用 GROUP BY 子句：

```
SELECT WORKDEPT, AVG(SALARY), MAX(SALARY)
  FROM EMPLOYEE
 GROUP BY WORKDEPT
```

如果我们只想列出超过 4 名员工的部门内的平均和最高薪水，就可以使用 HAVING 子句：

```
SELECT WORKDEPT, AVG(SALARY), MAX(SALARY)
FROM EMPLOYEE
GROUP BY WORKDEPT
HAVING COUNT(*) > 4
ORDER BY WORKDEPT DESC
```

容易混淆的两点注意事项：

- GROUP BY 子句与 ORDER BY 子句不同，前者的作用对象是查询的中间结果集。
- HAVING 子句只作用于组，从中选择满足条件的组。WHERE 子句中不能包含聚集函数，由于是在分组前被评估的，因此不能对分组执行任何函数。

8.2 搜索条件

8.2.1 谓词种类

在 ANSI/ISO 标准中，SQL 语句中的谓词是搜索条件的元素，用于明示或暗示比较操作。

DB2 支持丰富的搜索条件，允许我们准确有效地指定多种不同类型的查询，WHERE 子句的谓词主要有下列几种：基本谓词、量化谓词、BETWEEN、EXISTS、IN、LIKE、NULL、IS VALIDATED 和 XMLEXISTS。参见表 8-1 中常用的谓词和用途。

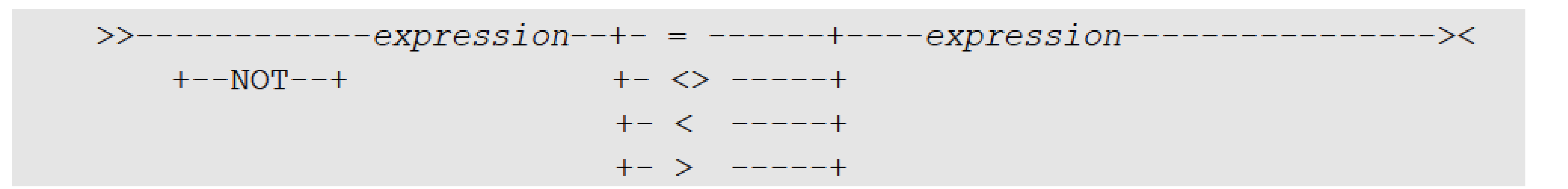
表 8-1 常用的谓词

查 询 条 件	谓 词
比 较	=、>、<、>=、<=、!=、<>、!>、!<
确定范围	BETWEEN AND、NOT BETWEEN AND
确定集合	EXISTS、IN、NOT IN
字符匹配	LIKE、NOT LIKE
空 值	IS NULL、IS NOT NULL
多重条件	AND、OR
XML 谓词	XMLEXISTS

8.2.2 基本谓词

基本谓词(basic predicate)用于对表达式的值和子查询生成的值进行简单比较。如果其中一个值是空值(NULL)，那么结果为未知的，否则结果为 TRUE 或 FALSE。

基本谓词在 SQL 表达式中的语法是：



```
>>------------expression--+- = ------+----expression----------------><
    +--NOT--+              +- <> -----+
                           +- <  -----+
                           +- >  -----+
```

```
                    +- <= -----+
                    '- >= -----'
```

列出员工号大于 300 的员工姓名、职业和薪水：

```
SELECT NAME, JOB, SALARY
  FROM STAFF
 WHERE ID > 300
```

我们也可以将几个基本谓词联合起来使用，列出员工号在 0 到 150 之间且不等于 100、部门号不为 50 的所有经理：

```
SELECT ID, NAME, JOB, DEPT
  FROM STAFF
 WHERE JOB = 'Mgr'  AND ID <> 100  AND ID >= 0  AND ID <= 150 AND NOT DEPT = 50
```

8.2.3　量化谓词

量化谓词(quantified predicate)是将单个值与一组值进行比较，基本语法为：

```
>>-+-expression1--+- = ------+--+-SOME-+--(fullselect1)---+---------><
   |              |             +-ANY--+                  |
   |              +- <> -----+  '-ALL--'                  |
   |              +- < ------+                            |
   |              +- > ------+                            |
   |              +- <= -----+                            |
   |              '- >= -----'                            |
   '-(------expression2---+--)--= --+-SOME-+--(fullselect2)-'
                                    '-ANY--'
```

列出员工号与任意某个部门号相同的员工信息：

```
SELECT *
  FROM STAFF
 WHERE ID = ANY (SELECT DEPT FROM STAFF)
```

在左边的表达式中，我们同样可以指定多个字段。比如，列出员工号与任意某个部门号相同，并且部门号与任意某个员工号相同的员工信息：

```
SELECT *
  FROM STAFF
 WHERE (ID, DEPT) = ANY (SELECT DEPT, ID FROM STAFF)
```

注意(ID，DEPT)的顺序不同，表示的查询结果也不相同。

8.2.4 BETWEEN、EXISTS 和 IN 谓词

BETWEEN 谓词将一个值与某一范围值进行比较，用法如下所示：

```
>>-expression--+-----+--BETWEEN--expression--AND--expression-------><
                '-NOT-'
```

列出年薪在 7 万到 8 万美元之间的员工信息：

```
SELECT *
  FROM STAFF
 WHERE SALARY BETWEEN 70000 AND 80000
```

需要注意的是，BETWEEN 后的值应该按照从小到大的顺序列出。

不同于 BETWEEN 谓词是用来指定某个范围，IN 谓词则用来检查表达式的值是否匹配由子查询生成的值中的一个。IN 谓词把某个数据值和由子查询产生的字段集合相比较，如果匹配字段中的某个值，就返回该值所处行的数据。

查询出部门号为 A00 和 B01 的员工信息：

```
 SELECT FIRSTNME, LASTNAME, WORKDEPT
 FROM   EMPLOYEE
 WHERE  WORKDEPT IN ('A00', 'B01')
```

EXISTS 谓词与 IN 谓词类似，只不过一般是与子查询出来的结果集进行比较。

比如下面例子中指定的结果集就是从新员工表(STAFFNEW)中查询出来的：

```
SELECT NAME, SALARY
FROM STAFF AS A
WHERE EXISTS (SELECT *
FROM STAFFNEW AS B
                WHERE B.ID = A.ID AND B.SALARY > 80000)
```

8.2.5 LIKE 谓词

LIKE 谓词用来做模糊匹配，主要使用的两个通配符是‘_’和‘%’，其中‘_’表示任意字符，‘%’表示 0 个或若干个字符。

```
 SELECT ID, NAME
 FROM   STAFF
 WHERE  NAME LIKE 'S%r'
 OR      NAME LIKE 'S%n'
 OR      NAME LIKE '__n%es'
```

从查询结果中可以看出：Sneider 对应'S%r'，Scoutten 对应'S%n'，而 Hanes、Jones 和 Gonzales 则对应'__n%es'。

```
50    Hanes
190   Sneider
200    Scoutten
260    Jones
320    Gonzales
```

8.2.6　NULL 谓词

NULL 谓词用来检查表中的空值，涉及空值查询时使用 IS NULL 或 IS NOT NULL。需要注意的是，这里的 IS 不能用等号(=)代替。

查询工作年限不为空值的员工姓名：

```
SELECT ID, NAME
FROM    STAFF
WHERE   YEARS IS NOT NULL
```

另外，在使用 ASC 排序时，NULL 值排在最大值的后面；反之在使用 DESC 排序时，NULL 值排在最大值的前面。表达式可以为 NULL，但不能等于 NULL，因此两个 NULL 值彼此不能判断为相等。

8.3　数据操作语言

比起查询语句，SQL 语言中的增删改操作要简单，但是在执行的时候一定要小心谨慎，也许一条不合理的 DELETE 语句会造成大量重要数据的丢失。本节简单介绍 INSERT、DELETE、UPDATE 和 MERGE 的语法和一些简单例子，在基本掌握后，我们可以套用基本语法来做各种复杂的数据操作。

8.3.1　INSERT

通过 INSERT 语句，我们可以向数据库的表、视图、昵称或全查询里插入数据，基本语法如下所示：

```
>>-INSERT INTO--+-table name-------+-------------------------------->
                +-view name--------+
                +-nickname---------+
                '-fullselect-------'
```

```
>--+-VALUES----+-+-expression-+-------------------------------------><
   |           | +-NULL-------+              |           |
   |           | '-DEFAULT----'              |           |
   '-+-------------------------------+--fullselect-'
     |                               |
     '-WITH----common-table-expression-+-'
```

其中，INTO 子句指定要插入数据的表名和属性列，VALUES 子句提供的值必须与 INTO 子句值的个数和类型匹配。如果 VALUES 子句中的值是通过子查询获得的，同样要匹配。

另外，在向目标对象里插入数据时还需要注意以下几点：

- 被选择的字段不允许包含字段函数，如 MAX、MIN 等
- 全查询中不允许包含 GROUP BY 和 HAVING 子句
- 插入的对象不允许包含连接(JOIN)

最简单的 INSERT 语句，就是向一张表(如 PRODUCTS)中插入一条记录：

```
INSERT INTO PRODUCTS (PRODID, PRODNAME, PRODLINE)
VALUES (8, 'Doll', 'Toys')
```

要向 PRODUCTS 表中插入两条产品记录，可用逗号隔开记录。如果其中 PRODLINE 的值分别为 NULL 或 DEFAUL，提交以后我们会发现 PRODLINE 的数据均为 NULL 值。

```
INSERT INTO PRODUCTS (PRODID, PRODNAME, PRODLINE)
VALUES (9, 'Pants', NULL),
        (10, 'Pills', DEFAULT)
```

当然，我们还可以根据搜索条件插入数据，比如向 EMPTEST 表中插入部门号为 D11 的员工信息：

```
INSERT INTO EMPTEST
       SELECT *
       FROM   EMPLOYEE
      WHERE   WORKDEPT='D11'
```

8.3.2 DELETE

与 INSERT 语句对应，DELETE 语句用来删除表、视图或全查询中的数据。

基本语法如下所示：

```
>>-DELETE FROM--+-table name-------+--------------------------------->
                +-view name--------+
                +-nickname---------+
                '-fullselect-------'
```

```
>-----------------------------------------------------------------><
                                        |                          |
                                        '-+-WHERE-search-condition-'
```

最简单的删除语句：

```
DELETE FROM EMPLOYEE
```

一般情况下，我们会通过 WHERE 子句删除表中的部分数据，例如删除所有女性员工和所在部门号为 C01 的员工信息：

```
DELETE FROM EMPLOYEE
      WHERE SEX='F' OR WORKDEPT = 'C01'
```

在 DELETE 语句中，如果 FROM 子句中包含全查询，最好先执行全查询中的 SELECT 语句，确认一下查询结果是否为我们想要删除的数据：

```
DELETE
FROM (SELECT *
       FROM EMPLOYEE
       WHERE JOB = 'MANAGER'
       ORDER BY SALARY DESC
       FETCH FIRST 5 ROWS ONLY)
```

执行完 DELETE 后，我们可以再次执行全查询中同样的 SELECT 语句，确认一下是否已经删除掉想要删除的数据。

8.3.3　UPDATE

UPDATE 语句可以用来更改数据库中表、视图或全查询数据项的值。基本语法如下所示：

```
>>-UPDATE-------+-table name-------+--------------------------------->
                +-view name--------+
                +-nickname---------+
                '-fullselect-------'
>--+-SET column name ----+-+-expression-+---------------------------><
                                        |                          |
                                        '-+-WHERE-search condition-'
```

其中，SET 子句指定要修改的列、修改方式和修改后的值，WHERE 子句指定修改条件。

最基本的 UPDATE 语句就是更新某张表中字段的值，比如给工作年限大于或等于 8 年的员工增加 10%的薪水：

```
UPDATE STAFF
SET     SALARY = 1.1*SALARY
WHERE   YEARS >=   8
```

在稍微复杂的 UPDATE 语句中，我们可以更新多个字段的值，并且在查询条件中使用 SQL 语言的字段函数。

比如把员工 SEAN 的工资和佣金调整至平均水平：

```
UPDATE EMPLOYEE E1
SET (E1.SALARY, E1.COMM)
=
(SELECT AVG(E2.SALARY), AVG(E2.COMM)
FROM EMPLOYEE E2
WHERE E2.WORKDEPT = E1.WORKDEPT)
WHERE E1.FIRSTNME = 'SEAN'
```

在执行 UPDATE 语句时，DB2 数据库会检查修改操作是否会破坏表上的完整性规则，比如用户自定义的 NOT NULL 和 UNIQUE 约束。

8.3.4 MERGE

MERGE 语句可以用来将一张表的数据合并到另外一张表中，在合并的同时可以进行插入、删除或更新等操作。

MERGE 语句的基本语法：

```
>>-MERGE INTO--+-table name-------+--------------------------------->
               +-view name--------+
               '-fullselect-------'
>--USING--------+-table name-------+--------------------------------->
                +-view name--------+
                '-fullselect-------'
>--ON-----+-search-condition -+-------------------------------------->
>-----WHEN MATCHED---------+-THEN-+-expression-+--------------------->
   |                       |                   |
   '-+-WHEN NOT MATCHED----+-THEN-+-expression-+--------------------->
>-------------------------------------------------------------------><
   '-+----ELSE IGNORE-------+-'
```

在使用 MERGE 之前，我们一定先要明确 MERGE 语句的 5 个主要要素：目标表、源表、匹配条件、匹配成功的操作和匹配失败的操作。在 MERGE 语句中根据判断条件，可以只选择 INSERT、UPDATE、DELETE 中的一种操作，也可以任意混合使用。

比如要将新员工表 STAFFNEW 中的数据合并到旧的员工表中，并更新员工的薪水：

```
MERGE INTO STAFF AS S                /* 目标表为 STAFF */
USING STAFFNEW SN                    /* 源表为 STAFF   */
ON S.ID = SN.ID                      /* 匹配条件       */
WHEN MATCHED THEN                    /* 匹配成功的操作  */
   UPDATE
   SET S.SALARY = SN.SALARY
WHEN NOT MATCHED THEN                /* 匹配失败的操作  */
   INSERT
   VALUES (SN.ID,SN.NAME,SN.DEPT,SN.JOB,SN.YEARS,SN.SALARY,SN.COMM)
```

在上面的 MERGE 语句中，5 个要素都齐全，其中匹配成功和失败的操作使用了 UPDATE 和 INSERT 语句，我们也可以仅仅使用 DELETE 语句删除两张表中匹配的数据：

```
MERGE INTO STAFF AS S
USING STAFFNEW SN
ON S.ID = SN.ID
WHEN MATCHED THEN
   DELETE
```

我们还可以在 MERGE 语句中使用全查询或字段函数，用来处理更复杂的业务逻辑，比如对员工号在 50 以内的数据做 DELETE 和 INSERT 操作：

```
MERGE INTO
    (SELECT * FROM   STAFF WHERE ID < 50
     )AS S
USING
     (SELECT * FROM STAFFNEW WHERE ID < 50
      )AS SN
ON    S.ID = SN.ID
WHEN MATCHED THEN
      DELETE
WHEN NOT MATCHED THEN
      INSERT
VALUES (SN.ID,SN.NAME,SN.DEPT,SN.JOB,SN.YEARS,SN.SALARY,SN.COMM)
```

相对于 INSERT、DELETE、UPDATE，MERGE 语句更加复杂。在执行之前，我们最好先分块检查一下 MERGE 语句的每个部分。

8.4 多表查询

在关系型数据库中，存放在不同表里的数据之间一般都有关联意义。相对于单表查询，多表关联查询应用更加广泛和复杂。多表的关联运算主要通过连接(JOIN)和集合运算来实现，经常使用的运算符参见表 8-2。

表 8-2 多表关联运算

关 联 方 式	谓 词
连 接	inner join、left outer join、right outer join、full outer join
集 合	union、union all、intersect、intersect all、except、except all、minus

下面简单介绍下这两种关联方式的用法。

8.4.1 JOIN 连接

连接是指根据数据的某些公共领域对来自两个或更多个表的数据进行组合的过程。如果连接条件(连接谓词)确定对应行中的信息匹配，那么一个表中的行就会与另一个表中的行配对。基本语法如下所示：

```
>>-SELECT--column name--FROM---+-INNER-------+-----------------------> 
                                +-LEFT----------------+
                                +-RIGHT--+    '-OUTER--'
                                '-FULL---'
>--+-JOIN table name—ON----join predicates-+----------------------><
                                           |                        |
                                           '-+-WHERE--predicates-'
```

ON 子句后的列名为连接字段，连接字段的类型不需要相同，但必须是可比较的。

比如员工号和对应的部门名称：

```
SELECT A.EMPNO, A.FIRSTNME, A.WORKDEPT, B.DEPTNO, B.DEPTNAME
FROM   EMPLOYEE A JOIN DEPARTMENT B
ON     A.WORKDEPT = B.DEPTNO
```

事实上，EMPLOYEE 表中的部门号(WORKDEPT)和 DEPARTMENT 表中的部门号(DEPTNO)表达的是同一意思，通过这两个字段我们可以关联查询两张表的信息。

在 DB2 里，默认的连接方式是内连接(INNER JOIN)，对应的其他方式称为外连接(OUTER JOIN)，两种方式的主要区别是：

- 内连接只显示满足连接条件的数据集。
- 外连接以指定表为连接主体，将主体表中不满足连接条件的数据集也显示出来

针对上面的例子，我们如果使用右外连接(RIGHT OUTER JOIN)的话，就会以表 DEPARTMENT 为主体表，显示出不满足条件的 NULL 值：

```
SELECT A.EMPNO, A.FIRSTNME, A.WORKDEPT, B.DEPTNO, B.DEPTNAME
FROM   EMPLOYEE A RIGHT OUTER JOIN DEPARTMENT B
ON     A.WORKDEPT = B.DEPTNO
```

部分查询结果如下所示，其中‘-’表示的就是 NULL 值：

```
EMPNO  FIRSTNME     WORKDEPT DEPTNO DEPTNAME
------ ------------ -------- ------ ------------------------------------
......
200340 ROY            E21       E21    SOFTWARE SUPPORT
-        -            -         H22    BRANCH OFFICE H2
-        -            -         J22    BRANCH OFFICE J2
-        -            -         G22    BRANCH OFFICE G2
-        -            -         D01    DEVELOPMENT CENTER
-        -            -         I22    BRANCH OFFICE I2
-        -            -         F22    BRANCH OFFICE F2
```

针对之前的内连接查询，使用 WHERE 子句代替 ON 子句可查询到相同的结果：

```
SELECT A.EMPNO, A.FIRSTNME, A.WORKDEPT, B.DEPTNO, B.DEPTNAME
FROM   EMPLOYEE A, DEPARTMENT B
WHERE   A.WORKDEPT = B.DEPTNO
```

虽然可以得到相同的结果，但是我们并不能使用 WHERE 子句替代 ON 子句的功能。因为两种方式实现的机制不一样，而且 WHERE 子句并不全适用于外连接的情况。为了防止查询语句书写混乱，我们只需要遵循如下原则：连接条件使用 ON 子句，过滤条件使用 WHERE 子句，桥梁功能和过滤功能不要混合使用。

8.4.2　集合运算

比起连接查询，集合运算主要用来处理数据之间的并(Union)、差(Difference)和交(Intersection)操作，相对应的 SQL 运算符为 UNION、EXCEPT 和 INTERSECT。

基本语法如下所示：

```
>>-SELECT--expression---+-UNION---------+------SELECT--expression----->
                        +-UNION ALL-----+
                        +-INTERSECT-----+
                        +-INTERSECT ALL-+
                        +-EXCEPT--------+
                        +-EXCEPT ALL----+
                        '-MINUS---------'
```

在这 7 种运算符中，UNION 和 UNION ALL 是使用最多的。

比如合并新旧两张员工表中的所有数据：

```
SELECT * FROM STAFF
UNION ALL
SELECT * FROM STAFFNEW
```

这里使用了 UNION ALL 运算符，DB2 会对两张表的所有数据做简单合并。但是，一般情况下我们不需要显示重复的数据，这时候就可以仅使用 UNION：

```
SELECT * FROM STAFF
UNION
SELECT * FROM STAFFNEW
```

下面我们分别创建测试表 TEST1 和 TEST2，并插入表 8-3 中的测试数据。

表 8-3　表 TEST1 和 TEST2 中的数据

TEST1	TEST2
A	A
A	A
A	B
B	B
B	B
C	C
D	

表 8-4 是通过几种运算符得到的对应结果：

表 8-4　表 TEST1 和 TEST2 的集合运算

运算符	结果
INTERSECT	A B C
INTERSECT ALL	A A B B C
EXCEPT	D
EXCEPT ALL	A D
MINUS	D

通过这些运算结果，我们就可以清晰地了解到每个运算符的作用。

8.5　高性能的 SQL 语句

在前面 4 节中我们主要讲解了 SQL 的基本语法和使用方法，在学会这些基础知识后，我们还需要将精力投入到书写高性能的 SQL 语句中。在实际应用中，一条糟糕的 SQL 语句可能影响到整个系统的良好运行。接下来我们将列出一些准则和注意事项，帮助书写出性能优良的 SQL 语句，提高 SQL 语句的运行速度。

8.5.1　高效 SQL 的准则

1. 高效的 SELECT 语句

因为 SQL 是一种灵活的高级语言，所以可以编写几种不同的 SELECT 语句来检索同一数据。但是，对于不同的语句形式和不同的优化类，性能可能相差很大。

请考虑下列书写高效 SELECT 语句的准则：

- 必须明确列出所选择的列的名称，不允许使用星号(*)代替。
- 尽量避免在 WHERE 子句的关联列中使用类型转换和函数运算。
- 尽量避免使用外连接，除非应用上要求必须使用。

- 合理地使用谓词关联，杜绝多表关联时出现笛卡尔积运算以及冗余谓词。如果 FROM 子句中包含 n 个表，那么 WHERE 子句中至少存在 n-1 个关联条件。
- 为了减少发生排序操作的可能性，避免出现不必要的 DISTINCT 、ORDER BY、GROUP BY、UNION 之类的子句。在不影响结果集的情况下，推荐使用 UNION ALL 和 EXCEPT ALL。
- 如果查询需要的行数远远小于结果集返回的行数(比如数据采样)，就必须使用 fetch first …only 或 optimize for n 子句。
- 使用游标时明确表达语句的目的，对于只读操作，使用 for read only 选项；对于更新操作，使用 for update 选项。
- 只读查询操作，绑定(bind/prep)时使用 blocking all 选项，利用行分块提高性能。
- 如果同一列出现在 OR 谓词中，就应该使用 IN 列表语句替代。

2. 最优化的 SQL 形式

SQL 语言的灵活意味着多种方法可以获取相同的正确结果。在查询执行期间，DB2 优化器将为每条 SQL 语句选择查询访问方案。DB2 优化器对许多备用访问方案的执行成本进行建模，并且将选择估算成本最低的访问方案。虽然 DB2 优化器具有强大的查询重写功能，但它并非始终能够将 SQL 语句重写成最优的形式。关于 DB2 优化器，我们在《DB2 数据库性能调整和优化(第 2 版)》的第 9 章“DB2 优化器”中详细讲解。

我们需要注意的是某些 SQL 构造会对查询 DB2 优化器所考虑的访问方案造成限制，您应该尽可能避免或替换这些构造。

- 尽量避免在搜索条件中使用复杂的表达式，这些表达式将导致优化器无法使用目录统计信息来估算精确的选择性。
- 如果对表达式使用连接谓词，那么连接方法只能是嵌套循环连接。
- 不要在局部谓词中使用基于列的表达式，而是使用表达式的翻转形式。
- 在某些情况下，数据类型不匹配将导致无法使用散列连接。如果连接列的数据类型是 CHAR、GRAPHIC、DECIMAL 或 DECFLOAT，那么长度最好相同。
- 尽量避免使用空操作表达式(coalesce)，DB2 优化器无法详细分析该谓词。
- 避免使用非等式连接谓词，主要是比较运算符不是等式的连接谓词，因为连接方法只能是嵌套循环连接。
- 避免使用在同一子查询中执行多次 DISTINCT 聚集操作的查询。如果无法避免使用多个 DISTINCT 聚集，那么请考虑将 DB2_EXTENDED_OPTIMIZATION 注册变量与 ENHANCED_MULTIPLE_DISTINCT 选项配合使用。

- 避免使用冗余的谓词，当它们跨不同的表出现时尤其如此。在某些情况下，优化器无法检测谓词是否冗余。这可能导致低估基数。
- 请考虑定义唯一约束、检查约束和引用完整性约束。这些约束将提供语义信息，使 DB2 优化器能够重写查询以消除连接、通过连接下推聚集、通过连接下推 FETCH FIRST n ROWS、除去不必要的 DISTINCT 操作以及执行许多其他优化。
- 使用用户自定义函数(UDF)代替查询中的复杂部分。

8.5.2　提高插入性能的准则

1. 优化索引、约束和触发器

精简索引数量，删除不必要的索引、约束和触发器，可以提高插入的性能。

优化索引结构，通常情况下，将相异值多的列放在复合索引的所有列的前面；特殊情况是，对于按照特定维度批量导入全新数据的情况，将此维度放在索引的第一列。比如按日新增每天的交易数据，将日字段放在组合索引的第一列。

2. 利用表追加模式

交易型系统中很少存在删除操作，但是会存在大并发、小数据量的插入操作，建议此类数据表利用表的追加模式(APPEND ON)提高数据插入的性能。ALTER TABLE 语句的 APPEND ON 选项指定将追加表数据，并指定不保留关于页中可用空间的信息。这样的表不能带有集群索引。对于只增大不减小的表，此选项能够提高性能。

谨慎使用此选项，此选项一般可以作为特殊时刻的临时措施，最好不作为一项长期的设置。

3. 使用 IMPORT 和 LOAD 工具替代

DB2 自带的工具 IMPORT 实际上是 INSERT 的封装，不仅效率高而且有更多其他有用的功能。类似地，可以使用 LOAD FROM CURSOR 或 LOAD FROM CLI 来替代等价的 INSERT 语句，性能上会有更大提升。

4. 修改可用空间控制记录数

在将数据插入到表中之前，DB2 通过插入搜索算法先检查可用空间控制记录(FSCR)，以查找空间足以存储新数据的页。但是，即使 FSCR 指示某页的可用空间足够，该空间也可能因为已被另一个事务中未落实的删除操作保留而不可用。

DB2 注册变量 DB2MAXFSCRSEARCH 指定将记录添加到表时要搜索的 FSCR 数目。默认情况是搜索 5 个 FSCR。修改此值使您能够在插入速度与空间复用之间进行平衡。使

用较大的值将优化空间复用。使用较小的值将优化插入速度。将值设置为 –1 表示强制数据库管理器搜索所有 FSCR。如果搜索 FSCR 时找不到足够的空间，那么数据将被追加到表的末尾。

8.5.3 复杂查询的准则

1. 限制表连接操作涉及的表的个数

对于数据库的连接操作，我们可以简单地将其想象为循环匹配的过程，每一次匹配相当于一次循环，每一个连接相当于一层循环，N 个表的连接操作就相当于 N-1 层的循环嵌套。

一般情况下，在数据库的查询中涉及的数据表越多，查询的执行计划就越复杂，执行的效率就越低，为此我们需要尽可能限制参与连接的表的数量。

2. 限制嵌套查询的层数

应用中影响数据查询效率的因素除了参与查询连接的表的个数以外，还有查询的嵌套层数。对于非关联查询，嵌套的子查询相当于使查询语句的复杂度在算术级数的基础上增长，而对于关联查询而言，嵌套的子查询相当于使查询语句的复杂度在几何级数的基础上增长。因此，降低查询的嵌套层数有助于提高查询语句的效率。

3. 灵活应用中间表或临时表

在对涉及较多表的查询和嵌套层数较多的复杂查询进行优化的过程中，使用中间表或临时表是优化、简化复杂查询的重要方法。通过使用一些中间表，我们可以把复杂度为 M*N 的操作转换为复杂度为 M+N 的操作，当 M 和 N 都比较大时(M+N <<M*N)，查询的复杂度被大大地降低。

4. 使用一些改写复杂查询的技巧

转换连接类型，最好保证连接类型和字段长度及精度的完全一致。

对于主查询中包含较多条件而子查询条件较少的表使用 EXISTS，对于主查询中包含较少条件而子查询条件较多的表使用 IN。

如果要查询的数据在表中所占的比例较大，可以考虑使用并行查询来提高查询的执行速度。

对于涉及巨大的表的连接的统计查询，由于可能会造成大量的排序统计工作，使得查询的速度变慢，此时可以考虑使用 SQL PL 将原来的复杂查询修改为多个小的查询。

8.5.4　索引的注意事项

合理使用正确的索引是提高 SQL 语句执行效率的关键因素，对查询性能的影响很大，合适的索引甚至可以成百倍地提高查询的性能。对索引的使用需要注意以下一些问题：

1. 过度索引

一般情况下，使用索引可以缩短查询语句的执行时间，提高系统的执行效率，但是要避免以下过度索引的情况出现。对表建立了过多的索引，从而造成维护索引所需要的时间超过使用索引所降低的时间。因此联机事务表上的索引，最多不要超过 5 个。尽量避免建立过大的索引，建议单个索引的列不要超过 5 个。

2. NULL 值

避免在索引中使用任何可以为空的列。NULL 值是系统中目前尚无法确定的值，不能用大于、小于、等于运算符来比较，对 NULL 值的处理只能用是与否来判定，所有的对 NULL 值的判定都会引起全表扫描，除非同时使用其他的查询条件。

3. 复合索引

复合索引是使用多个数据列的索引，第一个字段的数据区分度非常重要，也是影响联合索引效率的关键所在。可以将区分度高的字段或所有 SQL 语句都引用到的字段作为第一字段。

4. 索引的相关参数

创建索引时，一般使用 ALLOW REVERSE SCANS 参数，允许双向扫描。

对于有很多列的大表的查询，如果返回的列不是很多，可以在建立索引的时候使用 INCLUDE 关键字将非条件列包含在索引里，这样数据库可以直接通过扫描索引拿出列值，就不会再去对表进行 FETCH 操作了。

5. 谓词与索引

在 DB2 中存在 4 种类型的查询谓词：range-delimiting(范围定界)谓词、index-sargable(索引控制)谓词、data-sargable(数据控制)谓词、residual(保留)谓词。其中 sargable 是 Search ARGument 的缩写，表示在确定查询条件时使用某个参数进行搜索。这 4 种谓词放在 WHERE 子句中作为查询条件时，性能依次下降。

这些类别列示如下，按最佳性能到最差性能的顺序排列：

范围定界谓词→索引控制谓词→数据控制谓词→保留谓词。

6. 根据 SELECT 使用的列建立索引

建立索引是用来提高查询性能的最常用方法。对于特定的查询，可以为某个表出现在查询中的所有列建立联合索引，包括出现在 SELECT 子句和条件语句中的列。但简单地建立能覆盖所有列的索引并不一定能有效提高查询，因为在多列索引中列的顺序是非常重要的。这个特性是由于索引的 B+树结构决定的。一般情况下，要根据谓词的选择度来排列索引中各列的位置，选择度大的谓词所使用的列放在索引的前面，把那些只存在于 SELECT 子句中的列放在索引的最后。

7. 在需要被排序的列上创建索引

这里的排序不仅仅指 ORDER BY 子句，还包括 DISTINCT、UNION 或 GROUP BY 子句，它们都会产生排序操作。由于索引本身是有序的，在其创建过程中已经进行了排序处理，因此在应用这些语句的列上创建索引会降低排序操作的代价。这种情况一般针对没有条件语句的查询。如果存在条件语句，DB2 优化器会首先选择出满足条件的记录，然后才对中间结果集进行排序。对于没有条件语句的查询，排序操作在总的查询代价中会占有较大比重，因此能够较大限度地利用索引的排序结构进行查询优化。此时可以创建单列索引，如果需要创建联合索引，那么需要把被排序的列放在联合索引的第一列。

8.6 本章小结

本章我们讲解了 SQL 语言相关的一些基础知识，算是 DB2 SQL 语言的入门学习。作为关系数据库的核心语言，SQL 是数据库开发人员或数据库管理人员都需要熟练掌握的，并且要求能够举一反三，解决实际查询场景中的问题。从某种意义上说，SQL 很简单，因为每个接触过数据库的人都会使用；但从另一个角度来看，SQL 也很复杂，因为在复杂的系统中，要想编写出简洁、高效的 SQL 语句并不是一件很容易的事。

关于 SQL 语句调优，在《DB2 数据库性能调整和优化(第 2 版)》一书的“第 11 章：SQL 语句调代”中会有更加详细系统的讲解。



DB2 基本监控方法

在系统出现这样或那样的性能问题时，你怎么知道什么地方才是性能调整努力的焦点呢？这就需要对数据库进行性能监控。

DBA 在动手做性能调整的时候一定要制定明确详细的计划，在头脑里一定要有个现实的可测量的目标。否则，就会成为一次无意义的练习。在改进数据库性能的时候，你必须首先清楚哪里会是性能的瓶颈并且有相应的对策。这就是 DB2 性能监控工具的用武之地。

本章主要讲解如下内容：

- 监控工具概述
- 快照监视器
- 快照监控案例讲解
- db2pd 及监控案例
- 事件监视器及监控案例
- db2mtrk 及监控案例
- 性能监控元素以及常用监控指标
- 性能监控总结

9.1 监控工具概述

DB2 数据库给我们提供了很多监控工具，有快照监视器、事件监视器、db2pd 工具、db2mtrk、Activity Monitor 等。要进行数据库配置参数调整，就必须监控数据库以获得有关锁定、连接、缓冲池使用、表空间使用和内存使用等方面的信息。要采集资源使用的详尽信息，必须使用 DB2 数据库监控工具。可以从 DB2 客户机或 DB2 服务器上执行监控功能。要调用监控工具，可使用 CLP 命令、图形性能监控界面或调用监控用的 API 接口。

Snapshot Monitoring(快照监视器)提供在特定时刻有关数据库活动的信息，是数据库活动当前状态的图像。当拍下快照时，返回给用户的数据量由监视器开关决定。这些开关可以在 DBM 配置文件中或在会话级别上设置。

Event Monitoring(事件监视器)记录 DB2 数据库在一段时间内的数据库活动，允许用户采集一段时间内的信息，包括死锁、连接、SQL 语句。

监视功能的性能影响取决于对被监视事件的监控频率和对每个事件捕获的数据量。快照监视器和事件监视器可以根据监视器定义由 DB2 立即调用。DB2 性能监控工具负责捕获和返回系统信息：快照监视器、一个或更多个事件监视器以及 db2pd。快照监视器会让你获得在指定时间点的状态映像。事件监视器在指定时间段内获取监视器数据并且将它们记入文件(命名管道)或表。

db2pd 工具是 DB2 V8.2 以后版本提供的一种非常强大的监控工具，用于收集 DB2 实例和数据库的统计信息。与 Informix Dynamic Server 的 onstat 工具类似，db2pd 提供了 20 多个选项用于显示关于数据库事务、表空间、表统计信息、动态 SQL、数据库配置和其他很多数据库细节的信息。db2pd 命令可以检索多个领域的信息，并把结果保存到文件中。也可以在特定时间段内连续多次调用该工具，帮助你了解随着时间的变化数据库的变动情况。该工具可用于故障诊断、问题确定、数据库监控、性能调优和辅助应用程序的开发设计。

数据库性能监控工具使用这些监控要素的组合来获取监视数据并为每个要素的数据存储提供了几种选择。快照和事件监视器均给予用户自由选择在文件还是在数据库表中保存收集到的数据的权利，进而通过屏幕查看或者使用定制程序来处理。数据库系统监视器通过一些自描述的数据流来将监视数据返回到客户端应用程序。使用快照监视应用程序，可以调用适当的 API 来获得快照信息，然后处理返回的数据流。使用事件监视应用程序，需要事先准备用文件或表来接收数据，激活相应的事件监视器，然后按照刚才接收数据那样处理数据流。快照监视器主要用于收集那些在它控制下的某一特定时间点的 DB2 实例和一些数据库状态信息。快照对于确定数据库系统的实时状态是非常有用的。采用固定的时间间隔，它们能够提供出一些让你观察性能趋势走向和辨认潜在的问题范围的信息。快照监视通过在命令行处理器(CLP)中执行 get snapshot 命令来进行。虽然快照监视器信息有助于诊断问题范围，但收集数据经常会引起额外的处理负担。这些系统级调用的成本可能是昂贵的，其他副作用是增加内存的使用：快照监视器数据是收集和存放在内存(DBM 的 MON_HEAP_SZ 参数)中而不是在某些特定的表或外部文件中。为了有效减少收集快照监视器数据的过载需求的数量，DB2 推荐使用控制被收集数据的数量和类型的快照监视器开关。像其他基本开关一样，每个快照监视器都有开和关两种状态的设置。当快照监视器开关打开的时候，在这个开关控制之下的一些监视器要素的信息被收集起来，相反也是(一定数量的监视信息不受这些开关的控制，因此这些信息不管那些开关被设置成什么状态总会被收集，例如 timestamp 监视信息)。

9.2　快照监视器

快照监视器概述

在数据库管理器级别，监视器开关设置在数据库管理器配置参数中。要查看所有的监视器开关设置的设置选项，请使用 get dbm cfg | grep DFT_MON(在 Windows 中执行命令 get dbm cfg |find /i "dft_mon")。要启用或禁用数据库管理器级别的监视器开关设置，请使用 update dbm cfg 命令，并指定要更改的个别监视器开关。例如，以下命令关闭 DFT_MON_LOCK 监视器开关，从而终止锁监视器数据的收集：

```
db2 update dbm cfg using DFT_MON_LOCK off
```

每个连接至数据库的应用程序(会话)都有自己的监视器开关集，这些监视器开关与数据库管理器和其他应用程序(会话)无关。应用程序(会话)在连接至数据库时，从数据库管理器继承它们的监视器开关设置。要查看会话的所有监视器开关设置的设置选项，请使用 get monitor switches 命令。可以使用 update monitor switches 命令来更改会话的监视器开关设置。例如，下面的命令打开 LOCK 监视器开关，从而启用 SNAPSHOT_LOCK 快照表函数使用的监视器元素的收集：

```
db2 update monitor switches using LOCK on
```

Snapshot Monitor(快照监视器)以计数器的形式提供累计信息。快照信息以一种特殊的数据结构提供，这种数据结构可以通过应用程序发出快照来检查。由监视数据结构返回的数据根据表 9-1 中定义的开关来设置。这些开关可以在实例(DBM 配置)级别或应用程序会话级别(update monitor switches)打开或关闭。表 9-1 包括执行快照时提供的概要信息，还有由 Snapshot Monitor 提供的基本信息。

表 9-1　快照监视器开关参数

组　别	所提供的信息	监视器开关	DBM 参数
排序	所用堆的数目、溢出、排序性能	SORT	DFT_ MON_ SORT
锁定	保持锁定数目、死锁数目	LOCK	DFT_ MON_ LOCK
表	测量活动(读行、写行)	TABLE	DFT_ MON_ TABLE
缓冲区	读和写的次数，所用时间	BUFFERPOOL	DFT_ MON_ BUFPOOL
工作单元	开始时间、结束时间、完成时间	UOW	DFT_ MON_ UOW
SQL 语句	开始时间、停止时间、语句标识	STATEMENT	DFT_ MON_ STMP

在 DBM(实例)配置参数中设置默认的开关值，在重启实例后，将影响实例中的所有数据库。而且与数据库相连接的每个会话将继承在 DBM 配置中设置的默认开关值。

1. 查看监控开关设置

在某种程度上，由于快照监视器开关控制着当快照被打开时所能够收集到的信息的类型和数量，因此你应该在开始监视进程之前搞清楚哪些开关是打开的而哪些是关闭的。获得这些信息的最简单方法就是通过在 CLP 中执行 get monitor switches 命令。在多分区数据库环境下，基本语法是：

```
get  monitor  switches  < AT DBPARTITIONNUM  [PartitionNum] >
```

其中，PartitionNum 参数用来说明需要获取快照监视器开关参数状态的数据库分区。尖括号(<>)显示的参数是可选参量，而方括号([ ])里面的参数是必需的。为了获取和显示单独分区数据库的快照监视器开关的状况，可以执行 get monitor switches 命令。假定均使用默认设置，结果如例 9-1 所示。

例 9-1　运行 get monitor switches 命令的结果。

```
$db2 get monitor switches

          Monitor Recording Switches

Switch list for db partition number 0
Buffer Pool Activity Information  (BUFFERPOOL) = OFF
Lock Information                    (LOCK) = OFF
Sorting Information                 (SORT) = OFF
SQL Statement Information        (STATEMENT) = OFF
Table Activity Information          (TABLE) = OFF
Take Timestamp Information       (TIMESTAMP) = ON  08/23/2012 09:21:33.838195
Unit of Work Information              (UOW) = OFF
```

从上面可以看到，TIMESTAMP 这个快照监视器开关的状态是 ON，而其他的都是 OFF。在这个开关状态后面显示的是这个开关打开的精确日期和时间。

2. 改变开关设置

在知道了每个可用的快照监视器开关的当前状态以后，你就可以在开始监控之前去改变其中的一个或多个开关的设置，从而获得相应的监控信息。你可以通过改变相对应的数据库管理器配置参数(该设置在实例重启后依然有效)，或者调用应用程序级别的 db2MonitorSwitches() API 函数，或者执行 update monitor switches 命令，在会话级别修改快

照监视器开关的设置，可以设置的开关参见表 9-1，这个命令的基本语法如下：

```
update monitor switches using [[SwitchID] ON | OFF ,...]
```

其中，SwitchID 指明一个或多个需要改变状态的快照监视器开关(该参数可以是以下其中的一部分或全部：BUFFERPOOL、LOCK、SORT、STATEMENT、TABLE、TIMESTAMP 和 UOW)。要将 LOCK 和 SORT 快照监视器开关参数的状态设置为 ON(会话级别)，可以执行 update monitor switches using lock on sort on 命令。

3. 获取数据

当数据库被激活或者与数据库的连接被建立时，快照监视器自动地开始收集监视器数据。但是，在你能够观察被收集的数据之前，必须选取快照(快照看起来就像是当时在那个时间点上的监视要素的映像)。你可以通过调用 db2GetSnapshot() API 函数或者执行 get snapshot 命令来得到快照。例 10-2 指明了这个命令的基本语法，Database Alias 用来说明需要做快照监视器信息的数据库别名。

使用这两种方法收集的信息都存储于监视器要素中(有时被认为是数据要素)；每个要素被设计成存储指定类型的信息。 下面列出的是可利用的监视器要素名单：

计数器

用来保留活动或事件发生次数的累计值(例如，对于数据库的已经执行的 SQL 语句的总次数)。计数器数值的增长贯穿监视器的生命周期；而在许多情况下，有可能会重置计数器。

计量值

表明项目的当前值(例如，当前连接到数据库的应用程序的数量)。

与计数器值不同的是，计量值可以变高或变低；它们在任意被测量点的实时值通常取决于数据库活动的级别。

高水位值

表明指标在监视开始以后曾经达到的最大值或最小值(例如，util_heap_sz 使用的最大值)。

信息要素

提供所有监视活动执行的细节信息(例如缓冲池名称、数据库名称和别名、详细路径等)。

时间戳

表明活动或事件发生的日期和时间(例如，第一次连接数据库建立的日期和时间)。时间戳被看成从 1970 年 1 月 1 日开始消逝的秒和微妙的数量值。

时间要素

记录时间被花费于执行活动或事件的成本(例如进行排序操作的时间花费)。时间要素的值会以从活动或事件开始所流逝的秒和微秒的数量形式来表现。一些时间要素可以被重置。

使用 get snapshot 命令可以要求快照。在检查 Snapshot Monitor(快照监视器)的输出之前，让我们可以讨论怎样捕获快照信息。可以利用 CLP 界面去捕获数据库监视器快照。

拍快照时，有可能定义相关领域。当需要数据库监视时，通常都有明确的需求。因此，如果对于特定的数据库存在与并发性(锁定)行为有关的问题，那么可以规定锁定级别。如果用户关心访问数据库的所有应用程序，那么就应当规定应用程序的级别。如果 STATEMENT 开关被打开并在数据库级别拍了快照，那么就能捕获有关表动态 SQL 语句活动(INSERT/UPDATE/SELECT/DELETE)的信息。对于数据库中每个表的数据库活动信息也能够捕获。可以提供以下级别的快照：

- Database Monitor(数据库管理器)——捕获活动实例的信息
- Database(数据库)——捕获所有数据库或单个数据库的信息
- Application(应用程序)——捕获所有应用程序或单个应用程序的信息
- Table Space(表空间)——捕获数据库中各个表空间的信息
- Table(表)——捕获数据库中各个表的信息
- Lock(锁)——捕获使用了数据库的应用程序持有的各种锁的信息

例 9-2 get snapshot 命令的语法。

```
Db2 get snapshot for dbm
Db2 get snapshot for database on dbname
Db2 get snapshot for tablespaces on dbname
Db2 get snapshot for bufferpools on dbname
Db2 get snapshot for tables on dbname
Db2 get snapshot for locks on dbname
Db2 get snapshot for applications on dbname
Db2 get snapshot for dynamic sql on dbname
```

仅仅想得到在 SAMPLE 数据库中被应用程序保持的锁定的快照信息，可以执行 get snapshot for locks on sample 命令。该命令输出的监控信息类似于例 9-3 中的结果(需要注意

的是，这只是一个非常简单的例子，真正监视器返回的监视数据通常要比这个大得多)。

例 9-3　数据库锁定快照。

```
            Database Lock Snapshot

    Database name                                = BODI
    Database path                                = /db2/db2mdsc/NODE0000/SQL00001/
    Input database alias                          = BODI
    Locks held                                  = 2
    Applications currently connected                 = 49
    Agents currently waiting on locks                = 0
    Snapshot timestamp                            = 08/29/2012 10:49:00.353875

    Application handle                            = 57967
    Application ID                               = 197.0.66.81.60713.121227081732
    Sequence number                              = 05208
    Application name                              = db2jcc application
    CONNECT Authorization ID                        = RUNFRP
    Application status                            = UOW Waiting
    Status change time                            = 08/29/2012 10:48:59.070883
    Application code page                          = 1208
    Locks held                                  = 0
    Total wait time (ms)                           = 0

    Application handle                            = 1013
    Application ID                               = 197.0.66.81.58428.120706064631
    Sequence number                              = 00366
    Application name                              = db2jcc_application
    CONNECT Authorization ID                        = RUNFE
    Application status                            = UOW Waiting
    Status change time                            = 08/29/2012 10:48:39.656078
    Application code page                          = 1208
    Locks held                                  = 0
    Total wait time (ms)                           = 0
    略………
```

可以监控同一实例控制下所有活动数据库的数据库数据、应用程序数据、缓冲池活动数据、表空间数据、表数据、锁的活动(关于所有保持锁定的锁的信息)、动态 SQL 数据(在 SQL 语句缓存中的当时关于 SQL 语句的信息)。

在后面的内容中，快照监视可用作寻找 DBM 和 DB 配置参数的最优设置的一种方式。

4. 重置计数器

另一个监视器要素被称为计数器，用于保存运行期间具体活动或事件发生的次数的数量累积。典型的计数开始于快照监视器开关打开或与数据库的连接被建立(如果实例级别的监视器被使用，计数开始于应用程序第一次建立与实例控制下的数据库连接的时候)。计数一旦开始，就将一直继续，直到适当的快照显示器开关被关闭或直到所有数据库连接被终止。但是，有时候也可以在你没有改变一个或更多个快照显示器开关状态和没有终止和重建所有当前活动数据库连接的情况下将所有计数器置为零。在这种情况下，所有监视器的计数器可以通过执行 RESET MONITOR 命令归零。这个命令的基本语法是：RESET MONITOR ALL 或 RESET MONITOR FOR [DATABASE | DB] [DatabaseAlias]。[DatabaseAlias]指明想要将快照监视器的计数器归零的数据库别名。如果想要重置实例控制下的所有数据库快照监视器的计数器，可以切换到这个实例，然后执行 RESET MONITOR ALL 命令。另一方面，如果只是想要把与 SAMPLE 数据库相关联的快照监视器的计数器重置为 0，那么可以这么做——执行 RESET MONITOR FOR DATABASE SAMPLE 命令。记住，不能使用 RESET MONITOR 命令有选择性地对由快照监视器开关控制的特殊监视器组重置计数器。如果要单独重置某个监视器的计数器，就必须在将相应的快照监视器开关关闭之后，再打开或终止重建数据库连接。在命令行调用快照监视器只是调用快照监视器的一种方法，并且在有些时候命令行调用快照并不是很好的选择。下面将会简单介绍另外一些可行的调用快照监视器的技术。

9.3 利用表函数监控

DB2 从 V9.7 版本开始提供了很多表函数，通过这些表函数可以获得很多与数据库性能有关的信息，所以也可以通过使用这些表函数，代替 get snapshot 命令来收集这些监控数据。

快照表函数能够捕获的许多监视器元素都受控于监视器开关。如果快照表函数中的某些函数描述中提到特定监视器开关，就表明受控于该开关。

在 DB2 的早些版本中无法使用 SQL 来捕获数据，获取快照监视器数据的唯一方式就是执行 GET SNAPSHOT 命令或者从应用程序中调用与之对应的 API 函数。而在 DB2 UDB 8.1 中，可以通过引用 20 多个可用的快照监视器表函数来构造查询以收集快照数据。表 9-2 列出这些函数并指明了它们所能获取的具体快照信息。

在 DB2 V8 中能够使用 SQL 表函数捕获快照。这代表一种明显的改进，从而可以轻松地捕获并存储快照，以便快速且灵活地进行检索。

表 9-2 列出了部分性能相关的表函数，更多的表函数请查询 DB2 相关文档。

表 9-2　部分表函数列表

快照表函数	返回的信息
SNAPSHOT_DBM	数据库管理器信息
SNAPSHOT_DATABASE	数据库信息。只有当至少有一个应用程序连接至数据库时，才会返回信息
SNAPSHOT_APPL	连接至分区数据库的应用程序上有关锁等待的应用程序信息，这包括累积计数器、状态信息和最近执行的 SQL 语句(假定设置了语句监视器开关)
SNAPSHOT_APPL_INFO	每个连接至分区数据库的应用程序的常规应用程序标识信息
SNAPSHOT_LOCKWAIT	有关锁等待连接至分区数据库的应用程序的应用程序信息
SNAPSHOT_STATEMENT	有关连接至分区数据库的应用程序的语句的应用程序信息，这包括最近执行的 SQL 语句(假定设置了语句监视器开关)
SNAPSHOT_TABLE	由连接至数据库的应用程序访问的每个表的表活动信息，需要表监视器开关
SNAPSHOT_LOCK	数据库级别的锁信息，以及每个连接至数据库的应用程序在应用程序级别的锁信息，需要锁监视器开关
SNAPSHOT_TBS	数据库级别的表空间活动信息、每个连接至数据库的应用程序在应用程序级别的表空间活动信息以及连接至数据库的应用程序已访问过的每个表空间在表空间级别的表空间活动信息。需要缓冲池监视器开关
SNAPSHOT_BP	指定数据库的缓冲池活动计数器。需要缓冲池监视器开关
SNAPSHOT_DYN_SQL	来自用于数据库的 SQL 语句高速缓存的某个时间点语句信息

快照监视器数据组织

所有的快照表函数都返回一张监视器数据表，其中的每一行代表一个正被监控的数据库对象实例，而每一列代表一个监视器元素(监视器元素代表数据库系统状态的特定属性)。

捕获监视器数据快照

要使用快照表函数捕获直接访问的快照，请完成以下步骤：

(1) 连接至数据库，可以是你需要监控的实例中的任何数据库。要能够使用快照表函数发出 SQL 查询，你必须连接至数据库。例如：

```
db2 connect to sample
```

(2) 确定你需要捕获的快照类型，以及你需要监控的数据库和分区。除了收集这个信息之外，请打开任何可应用的监视器开关(通过检查表 9-2 中的快照表函数描述可以确定这一点)。

例如，如果想要捕获表活动数据的快照(使用函数 SNAPSHOT_TABLE)，那么将需要激活 TABLE 监视器开关：

```
db2 update dbm cfg using DFT_MON_TABLE on
```

(3) 使用期望的快照表函数发出查询。

例如，以下查询捕获有关当前已连接分区的 SAMPLE 数据库的表活动信息的快照：

```
db2 "select * from table(SNAPSHOT_TABLE('SAMPLE',-1)) as T"
```

快照表函数有两个输入参数：

- VARCHAR(255)，用于数据库名称。如果输入 NULL，就使用当前已连接的数据库名称。这个参数不能应用于只返回数据库管理器信息的快照表函数(例如 SNAPSHOT_DBM)。这样的快照表函数只有一个分区号参数。对于以下的快照表函数列表，如果输入 NULL 来表示使用当前已连接的数据库，那么将得到实例中所有数据库的快照信息：SNAPSHOT_DATABASE、SNAPSHOT_APPL、SNAPSHOT_APPL_INFO、SNAPSHOT_LOCKWAIT、SNAPSHOT_STATEMENT 和 SNAPSHOT_BP。
- SMALLINT，用于分区号。对于分区号参数，输入整数(0 和 999 之间的值)以对应需要监控的分区号。要捕获当前已连接分区的快照，请输入值－1 或 NULL。要捕获全局快照，请输入值－2。

下面的语法将会创建引用非数据库管理器级别表函数的查询：

```
SELECT * FROM TABLE ( [FunctionName]([DBName],[PartitionNum]) AS [CorrelationName]
```

在这里，FunctionName 用来说明所使用快照监视器的表函数；DBName 指明需要从那个数据库的快照监视器中收集数据； PartitionNum 说明需要从那个数据库分区的快照监视器中收集数据；CorrelationName 则是查询产生的结果数据集的名称。构造引用数据库管理器级别快照监视器表函数的查询语法也是一样的。不同的是：DBName 参数不使用。如果想要获取分区数据库环境里当前分区的快照监视器数据，可以将 PartitionNum 参数的值设置为－1；如果希望获取所有分区的快照监视器数据，可以设置为－2。同样，如果想获取当前连接数据库的快照信息，可以把 DBName 参数设定成空值，也可以使用一对空的单引号或者使用 CAST 操作，例如 CAST(NULL AS CHAR)；如果想要通过使用快照监视器的表函数 SNAPSHOT_LOCK 来抓取包含与应用程序关联的当前连接的数据库的锁定数据的快照信息，可以执行下面的语句：

```
SELECT * FROM TABLE (SNAPSHOT_LOCK (CAST (NULL AS CHAR), -1) AS LOCK_INFO
```

如果使用 SAMPLE 数据库(先前的例子)作为当前被连接的数据库，那么执行 GET SNAPSHOT FOR LOCKS ON SAMPLE 命令后，返回的信息将会与先前的那个非常的相似。

9.4 性能管理视图及案例

DB2 V9 之后版本提供了很多性能管理视图(这些性能管理视图类似 Oracle 数据库中以 v$开头的动态性能视图)，使用这些管理视图可以获得与表函数和快照类似的监控数据，表 9-3 列出了部分管理视图，这些视图是 DB2 V9.5 的相关视图，V9.1 的视图略有不同。需要注意的是，这些视图的模式名都是 SYSIBMADM。

同样，这些视图中能够获得哪些监控数据很多取决于监控开关的设置情况。

表 9-3 部分管理视图

视 图 名	模 式 名	描 述
APPLICATIONS	SYSIBMADM	数据库中运行的应用
APPL_PERFORMANCE	SYSIBMADM	每个应用中 rows selected 与 rows read 的比率
BP_HITRATIO	SYSIBMADM	缓冲池的命中率
BP_READ_IO	SYSIBMADM	缓冲池读的信息
BP_WRITE_IO	SYSIBMADM	缓冲池写的信息
CONTAINER_UTILIZATION	SYSIBMADM	表空间中容器的利用率信息
LOCKS_HELD	SYSIBMADM	当前获得的锁的信息
LOCKWAITS	SYSIBMADM	锁等待的信息
LOG_UTILIZATION	SYSIBMADM	日志利用率的信息
LONG_RUNNING_SQL	SYSIBMADM	执行时间最长的 SQL
SNAPAGENT_MEMORY_POOL SNAP_GET_AGENT_MEMORY_POOL	SYSIBMADM	代理级别的内存使用情况
SNAPBP SNAP_GET_BP_V95	SYSIBMADM	缓冲池的基本信息
SNAPDYN_SQL SNAP_GET_DYN_SQL_V95	SYSIBMADM	数据库中动态 SQL 的执行情况
SNAPLOCKWAIT SNAP_GET_LOCKWAIT	SYSIBMADM	锁等待的信息
SNAPSTMT SNAP_GET_STMT	SYSIBMADM	应用中 SQL 语句的执行情况

(续表)

视　图　名	模　式　名	描　　述
SNAPTAB SNAP_GET_TAB_V91	SYSIBMADM	表的信息
SNAPTAB_REORG SNAP_GET_TAB_REORG	SYSIBMADM	重组信息
SNAPTBSP SNAP_GET_TBSP_V91	SYSIBMADM	表空间信息
TBSP_UTILIZATION	SYSIBMADM	表空间的利用情况
TOP_DYNAMIC_SQL	SYSIBMADM	消耗资源最多的 SQL 语句信息

上面的表格中列出了一些主要的视图，这些视图以较好的方式将快照获得信息进行良好的组织，使获得一些性能的信息变得简单容易。

DB2 V9.7 提供了可直接通过 SQL 访问的新关系监视接口，从而增强了数据库系统、数据对象和程序包高速缓存的报告和监视，以帮助你快速找出可能导致问题的情况。

新接口会报告一些监视元素，这些元素提供有关下列各项的信息：在系统中完成的工作、数据对象(比如表、索引、缓冲池、表空间和容器)以及程序包高速缓存中的 SQL 条目。与现有系统监视器和快照接口相比，为 DB2 V9.5 创建的工作负载管理(WLM)表函数之类的新接口更有效，并且对系统的影响较小。

可使用下列表函数，通过 SQL 直接访问系统、活动和数据对象级别的监视信息。

1. 系统级别

- MON_GET_CONNECTION
- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_UNIT_OF_WORK
- MON_GET_UNIT_OF_WORK_DETAILS
- MON_GET_WORKLOAD
- MON_GET_WORKLOAD_DETAILS

2. 活动级别

- MON_GET_ACTIVITY_DETAILS

- MON_GET_PKG_CACHE_STMT
- MON_GET_PKG_CACHE_STMT_DETAILS(仅从DB2 V9.7修订包 1 开始才可用。)

3. 数据对象级别

- MON_GET_BUFFERPOOL
- MON_GET_CONTAINER
- MON_GET_EXTENT_MOVEMENT_STATUS
- MON_GET_INDEX
- MON_GET_TABLE
- MON_GET_TABLESPACE

DB2 V9.7 借助新的监视元素和基础结构，可使用 SQL 语句有效地收集监视器数据，以确定系统的特定方面是否正常工作并帮助诊断性能问题，同时产生合理的性能开销。通过新访问方法，可获取所需的所有数据而不使用快照接口。提高的监视详细程度允许更好地控制数据收集过程，从源中收集想要的数据。

收集与应用程序执行的工作有关的监视信息，并通过下列 3 个级别的表函数接口进行报告：

- 系统级别：这些监视元素提供有关系统中执行的所有工作的详细信息。监视元素访问点包括服务子类、工作负载定义、工作单元和连接。
- 活动级别：这些监视元素提供有关系统中执行的活动(在系统中执行的一小部分特定工作)的详细信息。可使用这些元素来了解活动的行为和性能。监视元素访问点包括个别活动以及数据库程序包高速缓存中的条目。
- 数据对象级别：这些监视元素提供有关由数据库系统在特定数据库对象(比如索引、表、缓冲池、表空间和容器)中处理的工作的详细信息，从而使你能够快速找出可能导致系统问题的特定数据对象问题。监视元素访问点包括缓冲池、容器、索引、表和表空间。

DB2 中共有数以百计的度量值可供选择，但收集全部这些度量值将生成相当多的数据，从而严重影响生产力。选择需要具有下列特性的度量值用于日常监控：

- 易于收集：不希望必须使用复杂或成本高昂的工具来执行日常监视，并且不希望监视活动对系统造成沉重负担。
- 易于理解：不希望每次查看度量值时都必须查找度量值的含义。

- 与系统相关：并非所有度量值在所有环境中都提供有意义的信息。
- 灵敏而不过分灵敏：度量值的变化应该指示系统中的实际变化，度量值不应自行波动。

下面列举一些日常监控中常用的监控指标以方便大家在日常工作中借鉴使用。

缓冲池命中率：缓冲池命中率是最基本的度量值之一，提供有关系统利用内存来避免磁盘 I/O 的效率的重要整体度量。对于 OLTP 环境而言，数据命中率达到 80%~85%或更高，索引命中率达到 90%~95%或更高，通常被认为是良好情况。当然，也可以使用缓冲池快照中的数据针对各个缓冲池计算这些命中率。虽然这些度量值通常很有用，但对于频繁执行大型表扫描的系统(例如数据仓库)而言，数据命中率通常相当低；原因在于，数据被读入缓冲池之后，在被逐出以便为其他数据腾出空间之前不再被使用。

可以使用如下语句来获取当前数据库的缓冲池命中率：

```
    select substr(bp_name,1,20) as bp_name, int ((1- (decimal(pool_data_p_reads) /
    nullif(pool_data_l_reads,0)))*100) as data_hit_ratio,
    int ((1-(decimal(pool_index_p_reads)/nullif(pool_index_l_reads,0)))*100)
    as index hit ratio, int
((1-(decimal(pool data p reads+pool index p reads)
    /nullif((pool data l reads+pool index l reads),0)))*100) as BP hit ratio,
    int ((1-(decimal(pool async data reads+pool async index reads)/
    nullif((pool async data reads+pool async index reads+direct reads),0)))
*100)
    as Async read pct, int
((1-(decimal(direct writes)/nullif(direct reads,0)))*100)
    as Direct_RW_Ratio from table (snapshot_bp ('sample', -1)) as snapshot_bp;
```

输出如图 9-1 所示。

BP_NAME	DATA_HIT_RATIO	INDEX_HIT_RATIO	BP_HIT_RATIO	ASYNC_READ_PCT	DIRECT_RW_RATIO
IBMDEFAULTBP	94	78	90	97	100
IBMSYSTEMBP4K	-	-	-	-	-
IBMSYSTEMBP8K	-	-	-	-	-
IBMSYSTEMBP16K	-	-	-	-	-
IBMSYSTEMBP32K	-	-	-	-	-

图 9-1　输出的部分截图

每个事务的缓冲池物理读写次数如下：

```
    (POOL DATA P READS + POOL INDEX P READS +
     POOL TEMP DATA P READS + POOL TEMP INDEX P READS) / TOTAL COMMITS

    (POOL_DATA_WRITES + POOL_INDEX_WRITES) / TOTAL_COMMITS
```

这些度量值与缓冲池命中率紧密相关，但用途略有不同。虽然可以考虑命中率的目标值，但每个事务的读写次数没有可能的目标。那么，为何要关心这些计算呢？这是因为磁盘 I/O 在数据库性能方面是极为重要的因素，所以最好从多个角度对其进行审视。此外，这些计算包括写操作，而命中率只涉及读操作。最后，难以单独确定 94%之类的索引命中率是否值得尝试改进。如果每小时只有 100 次逻辑索引读操作，并且其中的 94 次在缓冲池内完成，那么为了使另外 6 次避免转变为物理读而耗费时间并不明智。但是，如果伴随 94%索引命中率的统计信息表明每个事务都执行 20 次物理读(这可以进一步分为数据读和索引读以及常规读和临时读)，那么缓冲池命中率可能值得进行调查。

使用如下语句进行查询：

```
    select DBPARTITIONNUM, (POOL DATA P READS + POOL INDEX P READS +
POOL TEMP DATA P READS + POOL TEMP INDEX P READS) / (COMMIT SQL STMTS +
ROLLBACK SQL STMTS ) as AVG AP BP READS ,(POOL DATA WRITES +
POOL INDEX WRITES)/ (COMMIT SQL STMTS + ROLLBACK SQL STMTS ) as
AVG_AP_BP_WRITES from SYSIBMADM.SNAPDB;
```

读取数据库行数与选择数据库行数之比：

```
    ROWS_READ / ROWS_RETURNED
```

此计算指示为了查找符合条件的行而从数据库表中读取的平均行数。数值较小表明查找数据的效率很高，这通常表示正在有效地使用索引。例如，在系统执行许多表扫描，并且需要检查数百万行才能确定它们是否符合结果集要求的情况下，此数值可能非常大。另一方面，通过全限定唯一索引来访问表时，此统计值可能非常小。纯索引访问方案(不需要从表中读取任何行)不会导致 ROWS_READ 增大。

在 OLTP 环境中，此度量值通常不会大于 2 或 3，这表明大部分访问操作通过索引而非表扫描完成。此度量值是监视方案在一段时间内的稳定性的简单方法，此值意外增大通常表明不再使用索引，在这种情况下，应该进行调查分析。

使用如下语句进行查询：

```
    select ROWS_SELECTED / ( ROWS_READ + 1 ) from SYSIBMADM.SNAPDB;
```

排序操作的平均耗时、平均每个事务的排序耗时、已经排序的溢出百分比，可使用如下语句进行查询：

```
    排序溢出百分比
    SRTOVP = (Sort overflows * 100) / Total Sorts
    每次排序的平均耗时
    SRTMS = Total sort time (ms) / Total sorts
    每个事务排序操作的平均耗时
```

```
    SRTMSTX = Total sort time (ms) / (Commit statements attempted + Rollback
statements attempted)
    select DBPARTITIONNUM,( SORT OVERFLOWS * 100 / (TOTAL SORTS + 1)) as
SRTOVP,( TOTAL SORT TIME * 100 / (TOTAL SORTS + 1)) as SRTMS ,( TOTAL SORT TIME
* 100 / (COMMIT SQL STMTS + ROLLBACK SQL STMTS )) as SRTMSTX from
SYSIBMADM.SNAPDB;
```

锁等待的平均时间以及每个事务的平均锁等待时间，可使用如下语句进行查询：

```
    平均锁等待
    LCKMS = Time database waited on locks (ms) / Lock waits
    每个事务的平均锁等待时间
    LCKTX = Time database waited on locks (ms) / TX Count
    select DBPARTITIONNUM,LOCK WAIT TIME / ( LOCK WAITS + 1 ) as
LCKMS,LOCK WAIT TIME / (COMMIT SQL STMTS + ROLLBACK SQL STMTS + 1 ) as LCKTX
from SYSIBMADM.SNAPDB order by DBPARTITIONNUM;
```

语句执行时间和返回的数据量，可使用如下语句进行查询：

```
    SELECT SUBSTR(DETMETRICS.STMT TEXT, 1, 40) STMT TEXT,
           DETMETRICS.ROWS RETURNED,
           DETMETRICS.STMT EXEC TIME
    FROM TABLE(MON GET PKG CACHE STMT DETAILS(CAST(NULL AS CHAR(1)),
        CAST(NULL AS VARCHAR(32) FOR BIT DATA),
        CAST(NULL AS CLOB(1K)), -1)) AS STMT_METRICS,
        XMLTABLE (XMLNAMESPACES( DEFAULT
          'http://www.ibm.com/xmlns/prod/db2/mon'),
          '$DETMETRICS/db2_pkg_cache_stmt_details' PASSING
        XMLPARSE(DOCUMENT STMT_METRICS.DETAILS) as "DETMETRICS"
        COLUMNS "STMT_TEXT" CLOB PATH 'stmt_text',
            "ROWS_RETURNED" BIGINT PATH 'activity_metrics/rows_returned',
            "STMT_EXEC_TIME" BIGINT PATH 'activity_metrics/stmt_exec_time'
        ) AS DETMETRICS
    ORDER BY rows_returned DESC
    FETCH FIRST 10 ROWS ONLY;
```

输出信息与图 9-2 类似。

```
STMT_TEXT                                ROWS_RETURNED        STMT_EXEC_TIME
---------------------------------------- -------------------- --------------------
SELECT TABNAME, TABFORM, REFNAME FROM DD              4409720                21664
SELECT CREATE_TIME FROM SYSTOOLS.HMON_AT              3646875                67876
SELECT CREATOR, NAME, CTIME FROM SYSIBM.              3646875                 3010
SELECT STATS_TIME, INDEXTYPE FROM SYSIBM              2208566                51796
SELECT * FROM "T100" WHERE "SPRSL" = ? A              1846302                 4880
SELECT ATM.SCHEMA, ATM.NAME, ATM.CREATE_              1778394                27538
SELECT * FROM "TCP01" ORDER BY "CPCHARNO               687820                  611
SELECT * FROM "TSL1T" ORDER BY "SPRAS",                643905                  827
SELECT * FROM "ALCLASTOOL" WHERE "CUSGRP               614917                 2386
SELECT "TABNAME" FROM "D010TAB" WHERE "M               571314                  756

  10 record(s) selected.
```

图 9-2　输出信息

9.5　快照监视器案例

9.5.1　监控案例 1——动态 SQL 语句

例 9-4 中显示的脚本将发出"get snapshot for all on **dbname**"命令，该命令包括"get snapshot for dynamic SQL on **dbname** > **snap.out**"命令的所有输出。如果发现不会捕获很多的 SQL 语句，那么可以增加监控的时间。一条语句的输出的"Dynamic SQL Snapshot Result"部分看上去如例 9-4 所示。为了保证监控尽可能多的 SQL 语句，建议增加 DBM 配置参数 mon_heap_sz。

例 9-4　示例动态 SQL 快照。

```
   Dynamic SQL Snapshot Result
 Database name                      = SAMPLE
 Database path                      = C:\DB2\NODE0000\SQL00003\
        Number of executions          = 30
 Number of compilations               = 1
 Worst preparation time (ms)          = 1624
 Best preparation time (ms)           = 1624
 Internal rows deleted               = 0
 Internal rows inserted              = 0
 Rows read                   = 1230
 Internal rows updated               = 0
 Rows written                       = 0
 Statement sorts                     = 0
Total execution time (sec.ms)  = 0.934186
 Total user cpu time (sec.ms)         = 0.000000
 Total system cpu time (sec.ms)       = 0.000000
 Statement text                 = select * from sales
   ...
```

可以看到，在输出中可以搜索一些很有用的字符串。

"Number of executions"可以帮助找到应该调优的那些重要语句，对于帮助计算语句的平均执行时间也很有用。

对于执行时间很长的语句，单独执行一次或许对系统要求不多，但是累加起来的结果就会大大降低性能。应尽量理解应用程序如何使用该 SQL 语句，或许只需对应用程序逻辑稍微重新设计一下就可以提高性能。

看看是否可以使用参数标记，以便只需为语句创建一个包，这一点也很管用。参数标记可用作动态准备的语句(生成访问计划时)中的占位符。在执行的时候，就可以将值提供

给这些参数标记，从而使语句得以运行。

有时候，为了解决某些问题，使用"grep"(UNIX)和"findstr"(Windows)对快照输出执行初步的搜索非常方便。如果发现了什么东西，就可以通过打开快照输出并找到问题所在，以便作进一步调查。

例如，为了识别是否存在死锁，对于 UNIX，命令为：

```
grep -n "Deadlocks detected" snap.out | grep -v "= 0" | more
```

对于 Windows，命令为：

```
findstr /C:"Deadlocks detected" snap.out | findstr /V /C:"= 0"
```

再比如，要搜索执行得最频繁的语句，

对于 UNIX，命令为：

```
grep -n " Number of executions" snap.out | grep -v "= 0" | sort -k 5,5rn
| more
```

对于 Windows，命令为：

```
findstr /C:" Number of executions"  snap.out | findstr /V /C:"= 0"
```

注意：

在 Windows 平台上执行的时候，注意中英文转换，例如“Number of executions”在中文环境中应该用“执行数”代替。这里主要讲思路。

"Rows read"可帮助识别读取行数最多的 Dynamic SQL 语句。如果读取的行数很多，通常意味着要进行表扫描。如果这个值很高，也可能表明要进行索引扫描，扫描时选择性很小，或者没有选择性，这跟表扫描一样糟糕。

可以使用 Explain 工具来查看是否真的如此。如果有表扫描发生，那么可以对表执行一次 RUNSTATS，或者将 SQL 语句提供给 DB2 Design Advisor(db2advis)，以便令其推荐更好的索引，以此来进行弥补。如果是选择性很差的索引扫描，那么或许需要更好的索引。可以试试 Design Advisor。

```
grep -n " Rows read" snap.out | grep -v "= 0" | sort -k 5,5rn
findstr /C:" Rows read" snap.out | findstr /V /C:"= 0"
```

"Total execution time"是将语句每次执行时间加起来得到的总执行时间。可以很方便地将这个数字除以执行的次数，得到平均执行时间。如果发现语句的平均执行时间很长，那么可能是因为表扫描和/或出现锁等待。索引扫描和页面获取导致的大量 I/O 活动也是原

因之一。通过使用索引，通常可以避免表扫描和锁等待。锁会在提交的时候解除，因此如果提交得更频繁一些，或许可以弥补锁等待问题。

```
    grep -n " Total execution time" snap.out | grep -v "= 0.0" | sort -k 5,5rn
| more
    findstr /C:" Total execution time" snap.out | findstr /V /C:"= 0.0" |sort /R
```

"Statement text"显示语句文本。如果注意到了重复的语句，这些语句除了 WHERE 子句中谓词的值有所不同外，其他地方都是一致的，那么就可以使用参数标记，以避免重新编译语句。这样可以使用相同的包，从而帮助避免重复的语句准备，而这种准备的消耗是比较大的。还可以将语句文本输入到 Design Advisor(db2advis)中，以便生成最优的索引：

```
    grep -n " Statement text"  snap.out | more
    findstr /C:"Statement text"  snap.out
```

9.5.2　监控案例 2——通过表函数监控

以下示例将演示前面给出的各个步骤。在这个方案中，已有 3 个应用程序连接至 sample 数据库。两个是本地连接，另一个连接自远程客户机。其中一个本地应用程序已经对 sample 数据库的 STAFF 表中的记录做了一些更新。同时，远程应用程序已经在 sample 数据库的 SALES 表中插入了一条记录。另一个本地应用程序用于执行快照监控活动。

以下是这 3 个应用程序执行的命令和语句的顺序。

监控应用程序用于设置 DFT_MON_TABLE 监视器开关：

```
    db2 update dbm cfg using DFT_MON_TABLE on
```

应用程序 1(远程应用程序)用于向 SALES 表插入一条记录：

```
    db2 "insert into sales values ('03/20/2007','LEE','Atlantic',5)"
```

应用程序 2(本地应用程序)用于对 STAFF 表更新 12 条记录：

```
    db2 "update staff set salary = salary * 1.1 where JOB = 'Clerk'"
```

监控应用程序用于捕获数据库 sample 中表信息的快照：

```
    db2 connect to sample
    db2 "select * from table(SNAPSHOT_TABLE('SAMPLE',-1)) as T"
```

注意：

上面第 2 条语句中的数据库名字应为大写。

上述查询的结果集中包含许多列，因此从命令行读取会很困难。如果只对几个特定监视器元素感兴趣，那么可以将 select 语句限制在相关的监视器元素列。例如，以下就是这样一个查询及其对应的结果集。

监控应用程序：

```
    db2 "select snapshot timestamp, table name, rows written, rows read from
table(SNAPSHOT_TABLE('SAMPLE',-1)) as T"
```

结果如下所示：

```
    SNAPSHOT TIMESTAMP         TABLE NAME       ROWS WRITTEN   ROWS READ
    -------------------------- ---------------- ------------ ------------
    2012-04-07-09.33.27.468598 SYSROUTINES                 0            4
    2012-04-07-09.33.27.468598 STAFF                      12           47
    2012-04-07-09.33.27.468598 SALES                      1            0
    2012-04-07-09.33.27.468598 SYSTABLES                   0            2
    2012-04-07-09.33.27.468598 SYSPLAN                    0            1
    2012-04-07-09.33.27.468598 SYSEVENTMONITORS             0            1
    2012-04-07-09.33.27.468598 SYSDBAUTH                   0            5
    2012-04-07-09.33.27.468598 SYSBUFFERPOOLS               0            1
    2012-04-07-09.33.27.468598 SYSTABLESPACES               0            3
    2012-04-07-09.33.27.468598 SYSVERSIONS                 0            1
      10 record(s) selected.
```

存储定期捕获的监视器快照结果可以提供大量有用信息，从而确定数据库的状态和性能趋势。这样做的一种简单方式是对你正在监控的实例数据库中的监视器数据创建一个(或多个)表。例如，下面这个正在创建的表将要存储有关连接至实例中数据库的应用程序数的监视器数据。

监控应用程序：

```
    db2 "create table instance_snap (snap_time timestamp NOT NULL,
    local_cons bigint, rem_cons_in bigint)"
```

以下语句捕获实例中各个数据库的连接数的快照以及时间戳记，并将数据插入到上面创建的 INSTANCE_SNAP 表中。

监控应用程序：

```
    db2 "insert into instance_snap select snapshot_timestamp, local_cons,
    rem_cons_in from table (snapshot_dbm(-1)) as snapshot_dbm"
    db2 "select * from instance_snap"
    SNAP_TIME                LOCAL_CONS          REM_CONS_IN
```

```
    -------------------------- -------------------- --------------------
    2012-04-07-09.40.49.867659                    2                    1
      1 record(s) selected.
```

上述输出指出有两个本地应用程序和一个远程应用程序连接到了数据库 sample。

有了快照表函数和性能视图，就可以使用 SQL 轻松地捕获数据库系统监视器数据的快照。将所选监视器数据集存储到 SQL 表中的这种能力允许存在许多监控应用程序。定期捕获并存储数据库系统快照信息，对特定时间范围内数据库系统的使用情况和性能作统计分析。

9.5.3　编写快照监控脚本

在实际生活中，我们可能需要实时用快照监控数据库在某段时间内的活动，下面的脚本可以帮助我们在某个时间，每隔一定间隔来监控数据库的性能信息。

```
    @echo off
    REM
    REM take a snapshot after specified sleep period for a number of iterations
    REM parameters: (1) database name   ----输入数据库名称
    REM             (2) file name id-----输出文件
    REM             (3) interval between iterations (seconds)—监控间隔
    REM             (4) maximum number of iterations-----监控次数
    REM
    REM  Note: You may receive an error about the monitor heap being too small.
You may
    REM      want to set mon heap sz to 2048 while monitoring.---设置 mon heap sz
    :CHECKINPUT
    IF ""=="%4" GOTO INPUTERROR
    GOTO STARTPRG
    :INPUTERROR
    echo %0 requires 4 parameters: dbname filename id sleep interval iterations
    echo 举例 "getsnap.bat sample 0302 60 3"
    GOTO END
    :STARTPRG
    SET dbname=%1
    SET fileid=%2
    SET sleep_interval=%3
    SET iterations=%4
    db2 update monitor switches using bufferpool on lock on sort on statement
on table on uow on
    REM repeat the snapshot loop for the specified iterations
```

```
    SET i=1
    :SNAPLOOP
       IF %i% LSS 10 SET i2=0%i%
       IF %i% GTR 9 SET i2=%i%
       echo Starting Iteration %i2% (of %iterations%)
       db2 -v reset monitor all
       sleep %sleep_interval%
       db2 -v get snapshot for dbm > snap%i2%_%fileid%
       db2 -v get snapshot for all on %dbname% >> snap%i2% %fileid%
       echo Completing Iteration %i2% (of %iterations%)
       SET /a i+=1
       IF %i% GTR %iterations% GOTO ENDLOOP
       GOTO SNAPLOOP
    :ENDLOOP
    db2 update monitor switches using bufferpool off lock off sort off statement
off table off uow off
    db2 terminate
    :END
```

上面的脚本往往用在高峰期间出现性能问题时，通过运行该脚本能够帮助我们定位性能问题所在。

9.5.4 db2pd 及监控案例

db2pd 是用于监视各种 DB2 数据库活动以及故障排除的监控工具，是从 DB2 V8.2 开始随 DB2 引擎发布的独立实用程序，其外观和功能类似于 Informix onstat 实用程序(其实就是 IBM 收购 Informix 后，DB2 从 Informix 数据库那里借鉴过来的)。db2pd 是从命令行以一种可选的交互模式执行的。该实用程序运行得非常快，因为不需要获取任何锁，并且在引擎资源以外运行(这意味着甚至能在挂起的引擎上工作)。通过快照监视还可以收集 db2pd 提供的很多监视器数据，但 db2pd 和快照监视的输出格式却有很大不同。

1. 监控的例子

下面这些例子说明了如何用 db2pd 工具监控数据库环境。

例 9-5 如果希望了解当前 DB2 的级别和当前操作系统的信息，可以输入以下命令：

```
db2pd -version -osinfo
```

效果如图 9-3 所示。

```
Instance db2inst5 uses 64 bits and DB2 code release SQL09058
with level identifier 06090107
Informational tokens are DB2 v9.5.0.8, s110603, IP23263, Fix Pack 8.

Operating System Information:

OSName:   AIX
NodeName: ycghost3
Version:  6
Release:  1
Machine:  000D5F4AD400

CPU Information:
TotalCPU    OnlineCPU   ConfigCPU   Speed(MHz)  HMTDegree  Cores/Socket
64          8           64          3000        4           n/a

Physical Memory and Swap (Megabytes):
TotalMem    FreeMem     AvailMem    TotalSwap   FreeSwap
8192        1441        n/a         33280       33250

Virtual Memory (Megabytes):
Total       Reserved    Available   Free
41472       n/a         n/a         34691

Message Queue Information:
MsgSeg      MsgMax      MsgMap      MsgMni      MsgTql      MsgMnb      MsgSsz
n/a         4194304     n/a         n/a         n/a         4194304     n/a

Shared Memory Information:
ShmMax                ShmMin                ShmIds      ShmSeg
68719476736           1                     131072      0

Semaphore Information:
SemMap      SemMni      SemMns      SemMnu      SemMsl      SemOpm      SemUme      SemUsz      SemVmx      SemAem
n/a         131072      n/a         n/a         65535       1024        n/a         n/a         32767       16384

CPU Load Information:
Short     Medium    Long
1.682068  1.561630  1.101135

CPU Usage Information (percent):
Total     Usr       Sys       Wait      Idle
0.200000  0.000000  0.200000  0.000000  99.800000
```

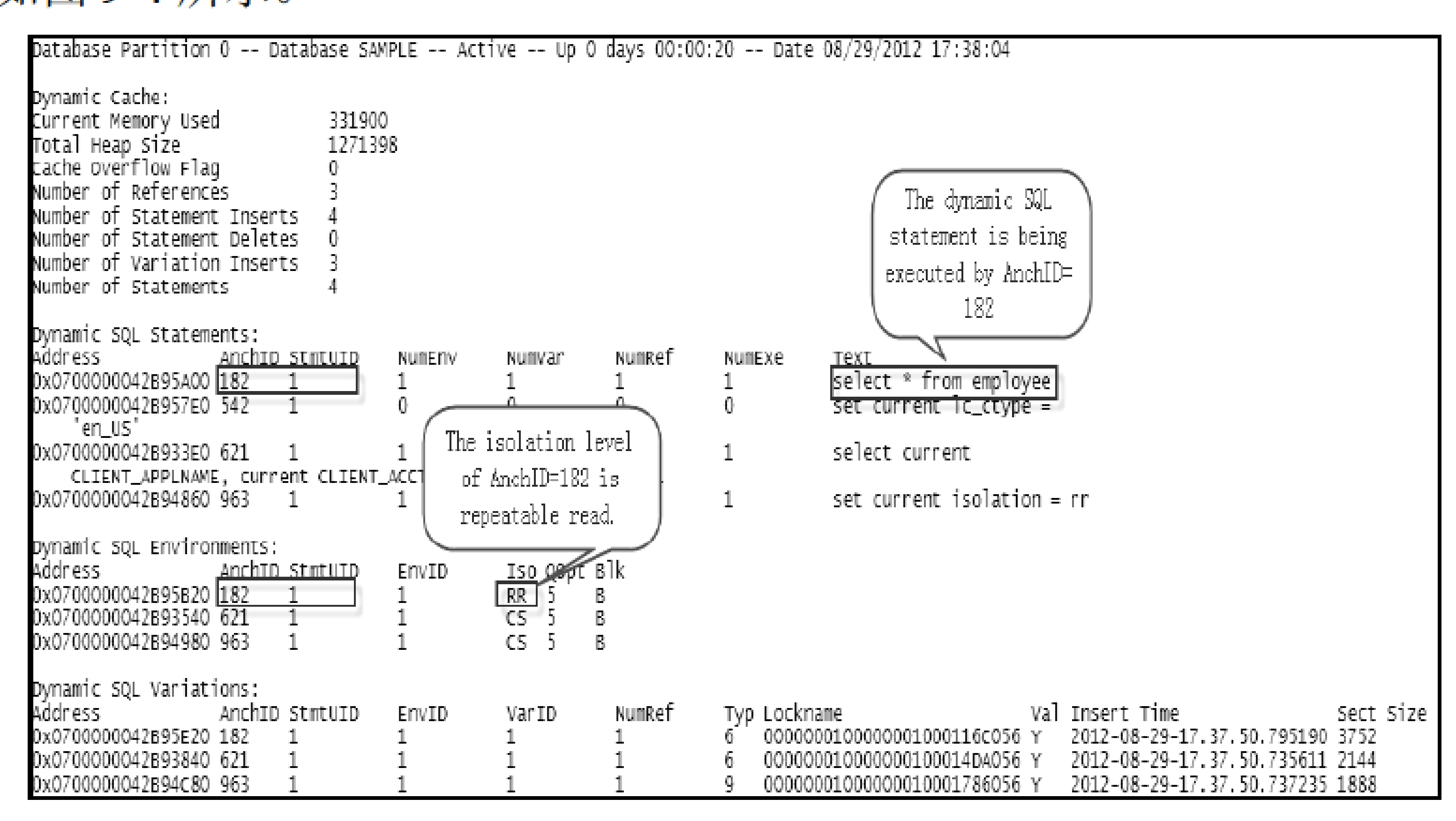
图 9-3　监控示例 1

–version 选项显示了系统中运行的当前 DB2 的版本和级别。输入 db2level 命令也可以得到同样的信息。–osinfo 选项显示 OS、CPU、物理内存和虚拟内存信息。类似的 OS 信息也可以在 DB2 启动时的 db2diag.log 中找到。这个例子也说明了获得版本信息和 OS 信息是多么简单，只需要在 db2pd 命令中指定两个选项即可。

例 9-6　如果需要检查动态 SQL 语句的当前隔离级别，可以使用下面的命令：

```
db2pd -db sample -dynamic
```

效果如图 9-4 所示。



```
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:00:20 -- Date 08/29/2012 17:38:04

Dynamic Cache:
Current Memory Used           331900
Total Heap Size               1271398
Cache Overflow Flag           0
Number of References          3
Number of Statement Inserts   4
Number of Statement Deletes   0
Number of Variation Inserts   3
Number of Statements          4

Dynamic SQL Statements:
Address            AnchID StmtUID    NumEnv     NumVar     NumRef     NumExe     Text
0x0700000042B95A00 182    1          1          1          1          1          select * from employee
0x0700000042B957E0 542    1          0          0          0          0          set current lc_ctype =
    'en_US'
0x0700000042B933E0 621    1          1                                1          select current
    CLIENT_APPLNAME, current CLIENT_ACCT
0x0700000042B94860 963    1          1                                1          set current isolation = rr

Dynamic SQL Environments:
Address            AnchID StmtUID    EnvID      Iso QOpt Blk
0x0700000042B95B20 182    1          1          RR  5    B
0x0700000042B93540 621    1          1          CS  5    B
0x0700000042B94980 963    1          1          CS  5    B

Dynamic SQL Variations:
Address            AnchID StmtUID    EnvID      VarID      NumRef     Typ Lockname                   Val Insert Time                Sect Size
0x0700000042B95E20 182    1          1          1          1          6   000000010000000100116C056  Y   2012-08-29-17.37.50.795190 3752
0x0700000042B93840 621    1          1          1          1          6   00000001000000010014DA056  Y   2012-08-29-17.37.50.735611 2144
0x0700000042B94C80 963    1          1          1          1          9   000000010000000100017860 56 Y   2012-08-29-17.37.50.737235 1888
```


图 9-4　监控示例 2

图 9-4 中，在 Dynamic SQL Environments 部分可以找到执行中的动态 SQL 语句的当前隔离级别。该例中，散列的锚标识符 182(**AnchID=182**)具有最严格的隔离级别——Repeatable Read(可重复读，RR)。通过交叉参照 Dynamic SQL Statements，可以确定哪条具体的 SQL 语句具有 RR 隔离级别：

```
select * from employee
```

当多个 DB2 用户并发地访问数据库时，锁等待会导致响应变慢。锁等待是临时性的，因而难以捕捉。然而，当出现锁等待情形时，需要由数据库管理员负责确定锁等待的原因。下面通过例子演示如何使用 db2pd 和 db2pdcfg 实用程序完成该任务。

2. 用于锁监视的 db2pd 选项

下面的图 9-5 展示了用于锁监视的 db2pd 选项。

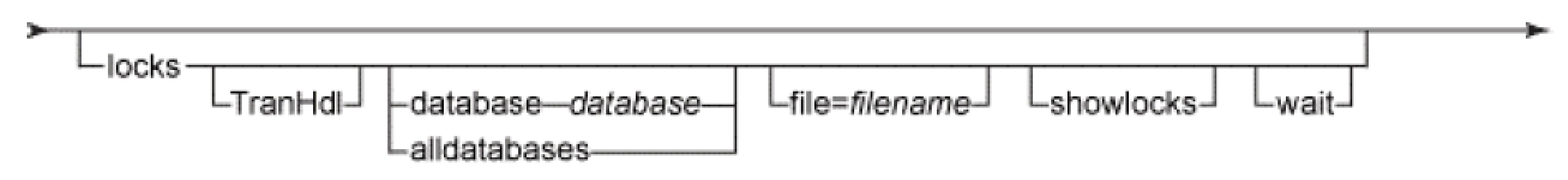


图 9-5　用于锁监视的 db2pd 选项

- TranHdl：用于指定事务句柄，以便只监视由特定事务持有的锁。
- showlocks：这个子选项将锁名扩展成有意义的解释。对于行锁，该选项显示以下信息：表空间 ID、表 ID、分区 ID、页和槽。通过使用编目视图 SYSCAT.TABLES 上的查询，很容易将表空间 ID 和表 ID 映射到相应的表名。

例 9-7　将表空间 ID、表 ID 映射到表模式、表名。

```
SELECT TABSCHEMA, TABNAME
FROM SYSCAT.TABLES
WHERE TBSPACEID = tbspaceid AND TABLEID = tableid
```

- wait：如果指定 wait 子选项，那么 db2pd 只显示事务当前正在等待的锁，以及对等待情形负责的锁。这个子选项大大简化了锁等待分析，因为能将输出限制为参与锁等待情形的锁。
- db2pd database 和 file 选项不是特定于锁监视的，但是适用于(几乎)所有 db2pd 调用。database 选项将 db2pd 返回的监视器数据限制为某个数据库的监视器数据。而 file 选项则允许定义一个文件，以便将 db2pd 输出写到该文件中。

3. 锁等待分析场景

用户 A 执行事务 A，根据每个经理的薪水为他们提供 10%的奖金。

事务 A 执行的更新操作如下：

```
 UPDATE EMPLOYEE SET BONUS = SALARY * 0.1  WHERE JOB = 'MANAGER'
```

当事务 A 仍然在运行(因为用户 A 还没有使用 COMMIT 或 ROLLBACK 终止该事务)时，用户 B 执行事务 B，将每个雇员的薪水提高 2%。

事务 B 执行的更新操作如下：

```
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
```

由于事务 B 没有完成，因此用户 B 请求 DBA 以确定问题产生的原因。于是，DBA 调用 db2pd，看是否存在锁等待情形：

```
    db2pd -db sample -locks wait showlocks
    Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:33:05
    Locks:
    Address    TranHdl    Lockname                  Type      Mode Sts Owner
Dur
    0x050A0240 6         02000600050040010000000052 Row     ..X  W   2          1
    0x050A0DB0 2          02000600050040010000000052 Row    ..X  G   2          1
    HoldCount  Att  ReleaseFlg
    0   0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
    0   0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
```

db2pd 报告 ID 为 2 的表空间中，在 ID 为 6 的表上有行锁存在锁等待情形。通过检查 SYSCAT.TABLES，DBA 断定表 EMPLOYEE 上的确存在锁等待。

确定锁等待情形涉及的表：

```
     SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABLES
    WHERE TBSPACEID = 2 AND TABLEID = 6
    TABSCHEMA                                TABNAME
    -----------------------------------------------------------------------
    ORACLE                                EMPLOYEE
      1 record(s) selected.
```

对于事务 2(列 TranHdl)，db2pd-locks 输出的 status 列(Sts)显示“G”。G 代表“granted”，意即事务句柄为 2 的事务拥有行锁。此外，列 Mode 表明，事务 2 持有的是 X 锁。等待的事务(列 Sts 中显示“W”(“wait”)的事务)是句柄为 6 的事务，该事务正在与事务 2 请求同一个行上的 X 锁。通过查看 Owner 列(显示事务 2 是锁的所有者)和比较 Lockname(对于

db2pd-locks 中的两个条目是相同的)，可以看到这一点。

接下来，DBA 将事务句柄映射到应用程序。这可以使用另一个 db2pd 选项-transactions 来完成：

```
db2pd -db sample -transactions
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:34:47
Transactions:
Address    AppHandl [nod-index] TranHdl    Locks     State   Tflag      Tflag2
0x05141880 30       [000-00030] 2          9        WRITE   0x00000000 0x00000
0x05144880 34       [000-00034] 6          5        WRITE   0x00000000 0x00000
```

这个 db2pd 调用的输出表明，事务 2(列 TranHdl)是由应用程序 30(列 AppHandl)执行的，而事务 6 是由应用程序 34 执行的。这两个事务都正在对数据库执行写更改(列 State=WRITE)。所以 DBA 现在知道，应用程序 30 正持有应用程序 34 等待的锁。

要获得关于锁等待情形涉及的应用程序的更多信息，可使用-agents 选项调用 db2pd。该选项打印代表应用程序运行的代理的信息。注意，-agents 是实例级选项，这意味着不需要指定数据库(实际上，当指定数据库时，db2pd 打印出一条警告并忽略 database 选项)。

获得关于应用程序和相应代理的信息：

```
db2pd -agents
Database Partition 0 -- Active -- Up 3 days 08:35:42
Agents:
Current agents:      2
Idle agents:         0
Active coord agents: 2
Active agents total: 2
Pooled coord agents: 0
Pooled agents total: 0
Address     AppHandl [nod-index] AgentTid   Priority   Type     State
0x04449BC0 34       [000-00034] 3392       0          Coord    Inst-Active
0x04449240 30       [000-00030] 2576       0          Coord    Inst-Active
ClientPid  Userid   ClientNm Rowsread   Rowswrtn   LkTmOt DBName
3916       USER B   db2bp.ex 43         43         NotSet SAMPLE
2524       USER_A   db2bp.ex 153        14         NotSet SAMPLE
```

在 db2pd-agents 输出中，DBA 可以看到使用应用程序 30 和 34 的用户的 ID(列 Userid)：应用程序 30 是由 USER_A 执行的，而应用程序 34 是由 USER_B 执行的。只有当每个用户都有单独的数据库授权 ID 时，才可能出现那样的应用程序与用户 ID 之间的映射。通常，这对于在应用服务器上运行的应用程序是不可能的，因为这些应用程序使用连接池，连接不是个人化的。

关于每个应用程序的更多信息则由 db2pd 选项-applications 提供：

```
db2pd -db sample -applications
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:36:14
Applications:
Address    AppHandl [nod-index] NumAgents  CoorTid    Status
0x04AF8080 34      [000-00024] 1         3940      Lock-wait
0x03841960 30      [000-00020] 1         2548      UOW-Waiting
C-AnchID C-StmtUID  L-AnchID L-StmtUID  Appid
195      1         0        0         *LOCAL.DB2.061122195637
0        0         60       1         *LOCAL.DB2.061122195609
```

Status 列确认了 DBA 已经知道的一些东西：应用程序 34 处在锁等待状态。但是这并不新鲜，于是 DBA 将注意力集中在列 C-AnchID/C-StmtUID 和 L-AnchID/L-StmtUID 上。“C”代表当前(current)，“L”代表最近(last)的锚 ID/语句 UID。这些 ID 可用于标识应用程序最近执行的 SQL 语句和应用程序当前执行的语句。为此，可以用-dynamic 选项调用 db2pd。该选项显示数据库动态语句缓存的内容：

```
db2pd -db sample -dynamic
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:37:39
Dynamic Cache:
Current Memory Used          187188
Total Heap Size             1271398
Cache Overflow Flag          0
Number of References         2
Number of Statement Inserts   3
Number of Statement Deletes   0
Number of Variation Inserts   2
Number of Statements         3
Dynamic SQL Statements:
Address    AnchID StmtUID   NumEnv    NumVar    NumRef    NumExe
0x056CEBD0 60     1         1         1         1         1
0x056CE850 180    1         0         0         0         0
0x056CFEA0 195    1         1         1         1         1
Text
UPDATE EMPLOYEE SET BONUS = SALARY * 0.1 WHERE JOB = 'MANAGER'
SET CURRENT LOCALE LC CTYPE = 'de DE'
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
Dynamic SQL Environments:
Address    AnchID StmtUID   EnvID Iso QOpt Blk
0x056CECD0 60     1         1     CS  5    B
0x056D30A0 195    1         1     CS  5    B
```

```
Dynamic SQL Variations:
Address    AnchID StmtUID    EnvID VarID     NumRef    Typ
0x056CEEB0 60     1         1     1         1         4
0x056D3220 195    1         1     1         1         4
Lockname
010000000100000001003C0056
01000000010000000100C30056
```

-applications 输出与-dynamic 输出之间的映射很简单：应用程序 34(处于锁等待状态)当前正在执行当前锚 ID 195 和当前语句 ID 1 标识的 SQL 语句。在 db2pd -dynamic 输出的 Dynamic SQL Statements 部分，那些 ID 可以映射到以下 SQL 语句：

```
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
```

持有锁的应用程序 30 最近执行的 SQL 语句是最近锚 ID 60 和最近语句 ID 1 标识的 SQL 语句。那些 ID 可以映射到以下 SQL 语句：

```
UPDATE EMPLOYEE SET BONUS = SALARY * 0.1 WHERE JOB = 'MANAGER'
```

注意，db2pd -dynamic 输出包含另外一些通常难以发现的有趣信息：Dynamic SQL Environments 部分的列 Iso 中显示了正被执行的动态 SQL 语句的隔离级别(UR = Uncommitted Read，CS = Cursor Stability，RS = Read Stability，RR = Repeatable Read)。

总结一下 DBA 就用户 B 的应用程序被挂起的原因，看看有什么发现：

- 挂起是由表 EMPLOYEE 上某个独占式的行锁导致的。
- 持有锁的事务属于用户 A 执行的某个应用程序，而用户 B 的事务正在等待那个锁。
- 两条有冲突的语句都是表 EMPLOYEE 上的 UPDATE 语句。

有了这些信息，DBA 就可以开始采取一些必要的步骤来解决锁等待状况，例如建议用户 A 终止事务，或者强制关闭用户 A 的应用程序。此外，可以采取措施避免将来出现那样的状况，例如检查是否创建最合理的索引，使之通过索引扫描快速提交事务并释放锁。

在这个示例场景中，db2pd 被连续执行数次，每次使用单独的选项。现实中不会出现这样的情况。相反，db2pd 只需调用一次，调用的同时使用前面介绍的所有选项。

分析锁等待情形所需的带有所有选项的单个 db2pd 调用：

```
    db2pd -db sample -locks wait showlocks -transactions -agents -applications
-dynamic     -file db2pd.out -repeat 15 40
```

产生的输出由针对每个选项的输出组成，各部分输出之间的顺序与各选项在 db2pd 调用中的顺序一致。而且，请注意 db2pd 调用最后的两个附加选项：

- -file 表明 db2pd 输出应该被写到文件中。在示例调用中，输出被写到文件 db2pd.out 中。
- -repeat 表明 db2pd 应该每隔 15 秒执行一次，共执行 40 次(每隔 15 秒执行一次，共执行 10 分钟)。每次执行的输出被附加到-file 选项指定的文件后面。

-file 和-repeat 选项对于在一段时间内监视数据库活动比较有用。对于锁等待分析，这两个选项可以帮助捕捉只存在一小段时间的锁等待情形。例如，如果数据库参数 LOCKWAIT 被设置为 20 秒，那么等待锁的事务在过了 20 秒的等待时间后会被回滚。为了捕捉那样的锁等待情形，db2pd 的时间间隔必须设置为比 20 秒更短的时间间隔，例如例子中的 15 秒。

9.5.5　事件监视器及监控案例

快照监视器监控的是数据库的实时数据，Event Monitor(事件监视器)记录某事件(event)或转变(transition)出现时某段时间内数据库活动的情况。事件监视器收集监视器数据，例如特定事件或事务的发生。因此，事件监视器提供了当事件或活动发生的时候，不能使用快照监视器监视时收集数据库系统监视器数据的方法。例如，假设想要捕获每当死锁周期发生时的监视器数据。如果对死锁的概念比较熟悉，那么应该知道数据库有被称为死锁检测器的特殊进程(db2dlock)，在后台安静地运行并且在预定的间隔(dlchktime)时间内“苏醒”，用于为死锁周期扫描当前正在锁定的系统。如果在死锁周期内被发现，死锁检测器将会随机选择、回滚并且终止涉及在此次周期中的任意事务。结果，被选择出来的那个事务将会接收到 SQL 错误代码(-911)，并且所有实际上已经获得的锁被释放以便剩下的事务能够继续执行。像这样一系列的事件信息不能被快照监视器捕获，当死锁出现时，DB2 将通过对多个事务中的某个事务发出 ROLLBACK(回滚)操作来解决死锁问题。有关死锁操作的信息不容易用 Snapshot Monitor 捕获到，因为在能够拍得快照之前，死锁可能已经被解决了。然后，事件监视器却可以捕获事件的重要信息，因为可以在死锁周期被检测到的瞬间被激活。这两种监视器的另外一个显著的不同是：快照监视器以后台进程的方式驻留，从数据库连接建立就开始捕获监视器数据；相反，事件监视器必须在使用之前专门建立和激活。几个不同的事件监视器可以共存，并且每个事件监视器只有在特定类型的事件或事务发生的时候才会被激活。

Event Monitor 类似其他的数据库对象，因为它们是使用 SQL DDL(数据定义语言)创建的。主要的差别是：Event Monitor 可以像 Snapshot Monitor 的开关那样被打开或关闭。

当创建 Event Monitor 时，必须声明要被监视事件的类型。当以下事件发生时，相应事件记录便被记录下来：

DATABASE——当最后那个应用程序与数据库断开时，记录下一个事件记录。

TABLES——当最后那个应用程序与数据库断开时，记录下每个活动表的事件记录。

DEADLOCKS——对于每个死锁，记录下事件记录。

TABLESPACES——当最后那个应用程序与数据库断开时，对每个活动表空间记录下事件记录。

CONNECTIONS——当应用程序与数据库断开时，对每个数据库连接事件记录下事件记录。

STATEMENTS——对于由应用程序发出的每条 SQL 语句(动态或静态)，记录事件记录。

TRANSACTIONS——当事务完成(COMMIT 或 ROLLBACK)时，为事务记录事件记录。

Event Monitor 的输出存放在目录或命名管道(pipe)中(从 DB2 V8 后可以存放在表中，这些事件监控的表可以自动生成)。当 Event Monitor 被激活时，管道和文件的存在将得到证实。如果 Event Monitor 的目标位置是命名管道，那么提示从管道读出数据是应用程序的职责。如果 Event Monitor 的目标位置是目录，数据流将被写入一系列文件中。这些文件顺序编号并且都有文件附加名 evt (例如 00000000. evt、00000001. evt 等)。当定义事件监视器时，规定 Event Monitor 事件文件的大小与数目。

事件监视器的创建方法和步骤如下：

(1) 创建 SQL Event Monitor，写入文件：

```
    db2 create event monitor evmname for eventtype write to file 'directory'
示例：db2 create event monitor SQLCOST for deadlocks,statements write to file
'/db2db/event'
```

(2) 激活事件监视器(确保有充足的可用磁盘空间)：

```
    $> db2 "set event monitor SQLCOST state = 1"
```

(3) 让应用程序运行。

(4) 取消激活事件监视器：

```
    $> db2 "set event monitor SQLCOST state = 0"
```

(5) 使用 DB2 提供的 db2evmon 工具来格式化 SQL Event Monitor 原始数据(根据 SQL 吞吐率可能需要数百兆字节的可用磁盘空间)：

```
    $> db2evmon -db DBNAME -evm SQLCOST > sqltrace.txt
```

(6) 浏览整个已格式化的文件，寻找显著大的成本数(是相当耗时的过程)：

```
    $> more sqltrace.txt
```

在实际的性能调优中，事件监视器并没有被广泛运用，这是因为过去事件监视器的输出只能写到文件或命名管道中，直到 DB2 V8 版本才可以写到表中，使得 DBA 可以利用

SQL 去读取；其次是因为 DB2 的事件监视器在监控期间会产生非常大的文件。

1. 事件监控器案例

下面我们举一个事件监视器存放在表中的例子：对于数据库管理员，调优数据库常常是一项挑战。调优应用程序是一种方法，但在大多数生产系统中，DBA 很少甚至不能更改源代码，因此限制了他们调优应用程序的能力。这在 DBA 使用第三方工具时尤为如此。所以，通常最有效的调优方法是解决问题的根源，即从 SQL 语句本身入手。通常通过查找哪些 SQL 语句消耗的资源最多来获得最佳性能，然后决定采取一定的措施来减少资源消耗。

通常，在第一次安装数据库时，会将其性能调至最优，但随着时间的流逝，一些常用的东西开始变得越来越慢。这在拥有大量数据的系统(例如决策支持系统)中尤为如此。用户开始看到诸如锁升级、全表扫描以及排序这样的因素造成性能下降，这些操作往往会迫使系统访问磁盘而不是访问内存。当出现这种情况时，最好检查一下 SQL，看看可以做哪些改进。

我们举这个事件监控器案例，旨在解决的问题是：针对通常的活动，调优正在用于最频繁访问数据库的 SQL，最消耗资源的 SQL。为了简化如何监控应用程序中 SQL 语句的问题，我们通过 DB2 的事件监视器，确定哪些 SQL 语句消耗的资源最多。

2. 利用表的方式

1) 运行事件监视器

首先，必须创建事件监视器，运行监视器来收集将要分析的数据。

打开新的 DB2 命令行处理器会话，然后执行以下 DB2 UDB 命令：

```
db2 => update monitor switches using statement on
db2 => create event monitor sql trace for statements write to table
创建完之后，我们可以看到，DB2 自动生成了下面这张表
$ db2 list tables for all | grep -i "stmt"
STMT SQL TRACE                ORACLE          T
2012-08-29-15.09.35.781000
db2 => set event monitor sql_trace state=1
```

2) 创建事件监视器

(1) 使上述会话一直处于打开状态，直到这些数据库活动完成。确保 STMT_SQL_TRACE 所在的表空间拥有足够的存储空间，在此时间有可能会产生非常大量的数据。表空间的大小取决于用户想要捕获的 SQL 语句的数目和事务量。

(2) 执行正常的数据库活动，直到想要监控的时段结束。这一监控阶段可以是问题产

生时期，也可以是通常的数据库高峰期间。

(3) 回到在步骤(1)中打开的会话，然后发出以下语句：

```
db2 => set event monitor sql trace state=0
db2 => terminate
```

3) 分析输出

由于已经在数据库表 STMT_SQL_TRACE 中存储了所需要的信息，因此可以查询该表以确定“讨厌”且耗时的 SQL 语句。

需要确定 4 类 SQL 语句。在执行以下查询以确定这些语句之前，在数据库配置参数中，至少要为应用程序堆大小(applheapsz)分配 256 个页面。

- 按照执行时间降序排列执行耗时最长的 SQL 语句。为了确定这些语句，使用下面的 SQL SELECT 语句：

```
select stmt_text, (stop_time-start_time) "ExecutionTime(sec)"
from stmt_sql_tracewhere stmt_operation not in (7, 8, 9, 19 )
order by decimal (ExecutionTime) desc
fetch first 10 rows only
```

- 按照频率降序排列执行次数最多的 SQL 语句。可以用下面这条查询来确定这些语句：

```
select distinct(stmt_text),count(*) Count from stmt_sql_trace
where operation not in (7,8,9,19)
group by stmt_text
order by count(*)desc
fetch first 10 rows only
```

- 按照 CPU 时间降序排列最耗 CPU 时间的 SQL 语句。用下面这条查询来确定这些语句：

```
select stmt_text ,user_cpu_time "UserCPU(sec)" from stmt_sql_trace
where operation not in (7,8,9,19)
order by usrcpu desc
fetch first 10 rows only
```

- 按照总排序时间降序排列排序时间最长的 SQL 语句。用下面这条查询找到这些语句：

```
select stmt_text ,total_sort_time "TotalSortTime(ms)" from stmt_sql_trace
```

```
where operation not in (7,8,9,19 )
order by decimal(total_sort_time) desc fetch first 10 rows only
```

捕获每一类中的 SQL 语句，并将它们放在 tune.sql 文件中。将下面这行插入到该文件中，这样可以更改工作负载中每条语句的执行频率：

```
--#SET FREQUENCY <x>
```

这里的<x>表示随后要执行 SQL 语句的次数。你的 tune.sql 文件类似于下面这样：

```
--#SET FREQUENCY 100
SELECT COUNT (*) FROM EMPLOYEE;
SELECT * FROM EMPLOYEE WHERE LASTNAME='HAAS';
--#SET FREQUENCY 1
SELECT AVG (BONUS), AVG (SALARY) FROM EMPLOYEE
GROUP BY WORKDEPT ORDER BY WORKDEPT;
```

将这些 SQL 语句复制到 tune.sql 之后，检查任何 SQL 语句的 WHERE 子句中是否具有参数标志符(?)。将参数标志符改为适当的数据类型值，以便在没有任何错误的情形下执行 SQL 语句。

为了确定哪些索引可能提高性能，按如下所示执行索引顾问程序：

```
$cd /sapr3/prod
$db2advis -d sample -i tune.sql -t 0 -o tuneidx.sql
```

推荐的所有索引将放置在文件 tuneidx.sql 中。编辑该文件，在文件开始处添加一条连接语句：

```
connect to sample user userid using password;
```

在该文件末尾添加下面这行：

```
terminate;
```

现在可以运行该文件以创建推荐的索引：

```
$db2 -tf tuneidx.sql -z tuneidx.log
```

其中，tuneidx.log 捕获 tuneidx.sql 的所有输出。

9.5.6　db2mtrk 及监控案例

db2mtrk 是用于在 DB2 数据库中进行内存跟踪的工具，可以用于查看实例、数据库、代理进程当前对内存的使用状态。

比如：

```
db2mtrk -i -d -v
 Memory for database:  SAMPLE
   Backup/Restore/Util Heap is of size 16384 bytes
   Package Cache is of size 81920 bytes
   Catalog Cache Heap is of size 65536 bytes
   Buffer Pool Heap is of size 4341760 bytes
   Buffer Pool Heap is of size 655360 bytes
   Buffer Pool Heap is of size 393216 bytes
   Buffer Pool Heap is of size 262144 bytes
   Buffer Pool Heap is of size 196608 bytes
   Lock Manager Heap is of size 491520 bytes
   Database Heap is of size 3637248 bytes
   Other Memory is of size 16384 bytes
   Application Control Heap is of size 327680 bytes
    Application Group Shared Heap is of size 57344000 bytes
   Total: 67829760 bytes
```

再比如：

```
$db2mtrk -i -d -p -r 1200
Memory for instance

   other      fcmbp      monh
   16.9M      832.0K     320.0K

Memory for database: SAMPLE

   utilh      pckcacheh   other      catcacheh   bph (1)     bph (S32K)
   64.0K      256.0K     192.0K     192.0K     8.2M       832.0K

   bph (S16K)  bph (S8K)   bph (S4K)   shsorth     lockh       dbh
   576.0K     448.0K     384.0K     0          640.0K     34.1M

   apph (19500)apph (19499)apph (19498)apph (19497)apph (19496)appshrh
   64.0K      64.0K      64.0K      64.0K      64.0K      192.0K

Memory for agent 8569

   other
   192.0K

Memory for agent 8312
```

```
    other
    192.0K

Memory for agent 8055

    other
    320.0K

Memory for agent 9033

    other
    576.0K

Memory for agent 8826

    other
    192.0K
```

db2mtrk 工具的语法如下：

```
>>-db2mtrk--+----+--+----+--+----+--+----+------------------------->
            '-i-'   '-d-'   '-p-'   +-m-+
                                    '-w-'
>--+-----------------------+--+----+--+----+--------------------><
   '-r--interval--+-------+-'  '-v-'  '-h-'
                  '-count-'
```

db2mtrk -i　　显示当前实例的内存使用情况

db2mtrk -i -v　　显示当前实例的内存使用情况的详细信息

db2mtrk -d　　显示数据库的内存使用情况

db2mtrk -d -v　　显示数据库的内存使用情况的详细信息

db2mtrk -p　　显示代理进程的专用内存使用率

db2mtrk -h　　显示帮助信息

-m 参数选项用于显示最大的内存使用上线

-w 参数选项用于显示使用过程中内存达到的最大值，即 water mark

-r 参数选项用于重复显示，其中的 interval 是重复显示的时间间隔数，count 是要重复显示的次数

interval 指定以秒为单位的重复的间隔

count 指定间隔的次数

-v 详细输出

-h 显示帮助信息

上面那个示例显示了实例级别和数据库级别内存使用情况的详细信息。

在 DB2 V9.1 中对该命令的-d 和-i 选项做了以下更改：

- 在 Windows 平台上，现在支持用于显示数据库级别内存的-d 选项。
- 由于现在可通过-d 选项来显示数据库级别内存，因此-i 选项仅显示实例级别内存。

在 db2mtrk 工具的输出信息中有下面几种类型的信息：

- 当前值的大小
- 最大值的限制(hard limit)
- 最高值(high water mark)
- 类型(用于指定使用了哪种内存)
- 代理进程使用的内容(只针对私有内存池)

在使用 db2mtrk 工具时，主要查看高水位和实际的配置之间是否接近。例如，如果 util_heap_sz 的高水位逼近了实际的数据库配置参数指定的值，那么说明你可以考虑增大该参数。

9.6　本章小结

本章详细说明了在 DB2 数据库中可以使用的监控工具和手段，针对具体的某些性能指标，可以通过上面提到的一个或多个工具进行监控。

在实际的生产环境中，在怀疑数据库中可能存在着性能问题的时候，就可以通过前面介绍的监控工具和手段来发现数据库中可能存在的瓶颈或不合理的资源使用。在实际发生性能问题的时候，问题产生的可能原因并不单一，而是由很复杂的原因造成的，所以往往需要通过多方面的监控分析才能得到准确的结论。因此，以上介绍的这些工具经常需要结合起来使用，然后对监控的结果综合进行分析，从而才能得到最准确的分析结论。

关于性能监控还有很多高级的内容，限于篇幅，我们就不在这里讨论了。如果读者还想更进一步地了解性能监控的知识，请参考《高级进阶 DB2(第 2 版)》和《DB2 数据库性能调整和优化(第 2 版)》。



运行数据库必须考虑的数据库设置

当我们按照前面几章的描述创建了数据库和相关的数据库对象之后，接下来要考虑的问题就是：如果数据库要作为生产数据库运行的话，应该至少设置哪些数据库参数或者调整哪些参数的默认配置，才能够满足生产数据库的要求。DB2 数据库中的参数有很多，足有几百个参数，像注册变量、实例配置参数(DBM CFG)、DB 配置参数(DB CFG)、表空间参数、表参数等，不胜枚举，而且随着业务系统运行一段时间之后，业务行为和业务量都在不断发生变化，很多原先设置的参数可能都需要进行调整，才能达到性能最优。本章不一一列举这些参数，着重介绍一下运行数据库必须考虑的数据库参数设置。

本章主要讲解如下内容：

- 数据库配置参数概述
- 通信设置
- 内存有关的设置
- 锁相关的设置
- 日志相关的设置
- 自动维护相关的设置
- 监控相关的设置
- 安全相关的设置

10.1　数据库配置参数概述

DB2 有几百个配置参数。很多参数都是由 DB2 自动配置(automatic)的，而其他一些参数都有它们自己的默认值，这些默认值都被证明在大多数环境中能够发挥得很好。接下来，我们只描述那些运行数据库常常需要重点关注的参数。

有些数据库配置参数可以在线更改(立即生效)，而另一些参数则要求对实例重启(执行 db2stop 之后接着执行 db2start)或数据库重启(断开数据库的所有连接或使用 deactivate 命令，再重新激活数据库)后才能生效。每个配置参数的文档都描述了该参数是否可以在线配置。

数据库管理器、数据库配置文件和 DB2 注册变量的基本管理命令如表 10-1 所示。

表 10-1　数据库管理器、数据库配置管理和 DB2 注册变量

命　　令	描　　述
GET DBM CFG [SHOW DETAIL]	列出数据库管理器配置文件中的当前值
UPDATE DBM CFG USING **config_param value**	将指定的数据库管理器配置参数设置成指定的值
GET DB CFG FOR **db_name** [SHOW DETAIL]	列出某个特定数据库的配置文件中的当前值
UPDATE DB CFG FOR **db_name** USING **config_param value**	将指定的数据库管理器配置参数设置成指定的值
db2set –lr	列出 DB2 已经设置的所有注册变量
db2set config_param=更改的值	将指定的 DB2 注册变量设置成指定的值

如果对配置参数做了更改，就可以用下面的 DB2 CLP 命令查看该设置是否立即生效(在线)：

```
GET DBM CFG SHOW DETAIL 和 GET DB CFG FOR dbname SHOW DETAIL
```

例如，在下面的输出结果中，MAX_QUERYDEGREE 和 MAXTOTFILOP 分别增加到了 3 和 19000。如果参数是可在线配置的，那么“Delayed Value”与“Current Value”应该是一样的。否则，就需要重新启动实例或者重新激活数据库。

```
Database Manager Configuration
Node type = Enterprise Server Edition with local and remote clients
Description                        Parameter     Current Value    Delayed Value
------------------------------------------------------------------------
 Maximum query degree of parallelism  (MAX QUERYDEGREE)  = 3              3
 Maximum total of files open            (MAXTOTFILOP)      = 16000         19000
```

在上面的例子中，对 MAXTOTFILOP 修改之后的值并未生效，这就需要重启实例之后才能生效。而对 MAX_QUERYDEGREE 修改之后的值已经生效了。

10.2　通信设置

1. DB2COMM

定义：

DB2COMM 注册变量允许您为当前 DB2 实例设置通信协议。如果 DB2COMM 注册变量未定义或设置为空，那么启动数据库管理器时不会启动任何协议连接管理器，应用程序通过非本地连接连接数据库就会失败。如果是通过 jdbc driver 连接数据库，就会返回类似如下的-4499 报错：

```
    An error occurred while establishing the connection:
    Type: com.ibm.db2.jcc.am.DisconnectNonTransientConnectionException
Error Code: -4499   SQL State: 08001
    Message:
    [jcc][t4][2043][11550][3.64.96] 异常 java.net.ConnectException：打开端口
60,004 上服务器 /197.3.137.249 的套接字时出错，消息为：Connection refused: connect。
ERRORCODE=-4499, SQLSTATE=08001
```

默认值：

NULL

是否可以联机配置：

否

设置指引：

更改该变量的设置，需要重启实例才能生效，生产数据库一般设置为 tcpip。

从 DB2 命令窗口输入 db2set DB2COMM 命令：

```
db2set DB2COMM=tcpip
```

2. svcename

定义：

此参数指定数据库服务器将用于等待来自远程客户机节点的通信的 TCP/IP 端口的名称。

默认值：

NULL

是否可以联机配置：
否

设置指引：
为了使用 TCP/IP 接收来自数据库服务器运行时客户机的连接请求，数据库服务器必须侦听指定给该服务器的端口。应将 svcename 参数设置为与主连接端口相关的端口号或服务名称，以便当启动数据库服务器时，可以确定在哪个端口上侦听入局连接请求。一般在生产数据库上，svcename 均设置为服务名称。服务名称和端口的对应关系可以通过查看/etc/services 文件获得。

更改该变量的设置，需要重启实例才能生效。

例如，设置服务名称：

```
db2 update dbm cfg using svcename DB2_instance
```

再如，设置端口号：

```
db2 update dbm cfg using svcename 60000
```

3. 节点配置文件 db2nodes.cfg

定义：
节点配置文件(db2nodes.cfg)位于实例所有者的主目录中，其中包含一些配置信息，告诉 DB2 数据库管理器有哪些服务器参与分区数据库环境的实例。

默认值：
当创建实例时，会自动创建 db2nodes.cfg 文件并对拥有实例的服务器添加条目。

是否可以联机配置：
否

设置指引：
这个文件对于单分区数据库不需要做任何改动，对于 DPF 数据库，需要手工编辑该文件以添加对应的条目。例如，如果正在对分区配置(有 4 台计算机，每台计算机上安装一个数据库分区服务器)进行安装，那么已更新的 db2nodes.cfg 应该类似于以下内容：

```
0     ServerA     0
1     ServerB     0
2     ServerC     0
3     ServerD     0
```

10.3　内存有关的设置

1. instance_memory

定义：
实例内存，此参数指定可以为数据库分区分配的最大内存量。

默认值：
automatic

是否可以联机配置：
是，可以动态更新。

设置指引：
该值可以保持为 automatic，在激活数据库分区时计算值。计算值介于物理内存的 75%到 95%之间，对于非独立的数据库服务器，根据实际需求指定大小。为了不过量使用物理内存，一般建议将该值设置为物理内存的 60%左右。

例如：

```
db2 update dbm cfg using instance_memory  XXXXX
```

使用 db2pd -dbptnmem 可以查看数据库内存的统计信息，通过下面的例子输出，我们可以知道实例用了 28553984KB 内存，其中大部分(28075264KB)用在了数据库 IBPS 上。

```
$db2 get dbm cfg |grep -i mem
Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = 8388608

=>这里我们将 instance_memory 设置为固定值：8388608*4KB=33554432KB
$db2pd -dbptnmem
Database Partition 0 -- Active -- Up 3 days 09:29:28 -- Date
2013-01-29-10.42.20.583112
Database Partition Memory Controller Statistics

Controller Automatic: N          =>N 代表 instance memory 不是 automatic 设置
Memory Limit:          33554432 KB   =>代表 instance memory 的值
Current usage:         28553984 KB   =>代表 DB2 当前耗用的内存量
HWM usage:           29694208 KB =>代表自上次启动以来，DB2 曾经使用过的最大内存量
Cached memory:        3745792 KB    =>未使用但为了提高将来内存请求的性能而高速缓存
的内存量
=>下面说明的是内存的使用分布情况
```

```
Individual Memory Consumers:

Name             Mem Used (KB) HWM Used (KB) Cached (KB)
========================================================
APPL-IBPS             160000        160000      134720  =>应用程序内存
DBMS-db2ibps           52608         52608       13760  =>全局数据库管理器内存
FMP_RESOURCES       22528         22528           0     =>与 FMP 通信所需的内存
PRIVATE               242816        360384       42048  =>其他专用内存
DB-IBPS             28075264      29398016     3555264  =>数据库 IBPS 消耗的内
                                                         存，参考 database_memory
LCL-p2883636             128           128           0  =>与本地应用程序通信的
                                                            内存段
LCL-p4260166             128           128           0
LCL-p3539134             128           128           0
LCL-p7274992             128           128           0
LCL-p5177416             128           128           0
LCL-p4915694             128           128           0
```

2. SELF_TUNING_MEM

定义：

此参数确定内存调整器是否根据需要在启用了自调整功能的内存使用者之间动态地分配可用内存资源。

默认值：

单分区环境：ON

多分区环境(DPF)：OFF

是否可以联机配置：

是

设置指引：

从 DB2 V9 开始，DB2 引入了内存自动调优的功能，通过自动设置几个内存配置参数值的方式，简化了内存配置任务。内存自动调优一旦启用(首先把该参数设置为 ON，其次把 database_memory 设置为 automatic，并且至少有两个内存使用者参数设置为 automatic)，就能够在内存使用者(包括排序堆、程序包缓存、锁定列表和缓冲池等)之间进行内存资源的动态分配。所以该参数在运行数据库上一般保持默认值 automatic。

3. database_memory

定义：
此参数指定为数据库共享内存区域保留的内存量。

默认值：
automatic

是否可以联机配置：
是

设置指引：
该参数代表数据库的共享内存，建议保持为 automatic，仅当启用了数据库的自调整内存功能(self_tuning_mem 配置参数设置为 ON)时，才会自动调整此配置参数。当内存调整器处于启用状态时，可确定数据库的整体内存需求并根据当前数据库需求增大或减小分配给数据库共享内存的内存量。例如，如果当前数据库需求很高，并且系统上有足够的可用内存，那么数据库共享内存将消耗较多的内存。当数据库内存需求下降时，或者当系统上的可用内存量降至过低水平时，就会释放一些数据库共享内存。

4. 缓冲池大小

定义：
缓冲池是内存中的一块存储区域，用于临时存放读入的数据页和将要写入的数据库页(包含表行或索引项)。缓冲池的用途是为了提高数据库系统的性能。从内存访问数据要比从磁盘访问数据快得多。因此，数据库管理器需要从磁盘读取或写入磁盘的次数越少，性能就越好。之所以配置一个或多个缓冲池是调优的最重要方面，是因为连接至数据库的应用程序的大多数数据(不包括大对象和长字段数据)操作都在缓冲池中进行，所以合理地设置缓冲池就至关重要了。

默认值：
在默认情况下，应用程序使用数据库在默认的缓冲池 IBMDEFAULTBP，这个缓冲池是在创建数据库时创建的。当 SYSCAT.BUFFERPOOLS 目录表中该缓冲池的 NPAGES 值为－1 时，DB2 数据库配置参数 BUFFPAGE 控制着缓冲池的大小。否则会忽略 BUFFPAGE 参数，并且用 NPAGES 参数指定的页数创建缓冲池。

是否可以联机配置：
是

设置指引：

缓冲池的默认配置一般无法满足生产数据库的要求，需要我们根据实际情况进行调整，调整的内容有两个方面：首先，调整缓冲池的个数；其次，调整每个缓冲池的大小。

为了判断缓冲池的效率，需要计算缓冲池命中率(BPHR)。理想的 BPHR 应超过 90%。公式如下：

```
    BPHR (%) = (1 - (("Buffer pool data physical reads" + "Buffer pool index
physical reads") /("Buffer pool data logical reads" + "Buffer pool index logical
reads"))) * 100
```

在 DB2 V9 中，可以使用 SYSIBMADM.BP_HITRATIO，也可以使用以下 SQL 语句来获得缓冲池命中率：

```
    db2 "select snapshot timestamp, substr(db name,1,10) as
dbname,substr(bp name,1,18) as bufferpool, total hit ratio percent as
total,data hit ratio percent as data, index hit ratio percent as index
    from sysibmadm.bp hitratio"

    示例输出：
    SNAPSHOT TIMESTAMP          DBNAME     BUFFERPOOL         TOTAL   DATA    INDEX
    -------------------------- ---------- ------------------ ------- ------- -------
    2013-01-29-16.15.29.883314 IBPS        IBMDEFAULTBP       95.17   93.92   96.85
    2013-01-29-16.15.29.883314 IBPS        IBMSYSTEMBP4K          -       -       -
    2013-01-29-16.15.29.883314 IBPS        IBMSYSTEMBP8K          -       -       -
    2013-01-29-16.15.29.883314 IBPS        IBMSYSTEMBP16K         -       -       -
    2013-01-29-16.15.29.883314 IBPS        IBMSYSTEMBP32K         -       -       -

      5 record(s) selected.
```

这个计算考虑了缓冲池高速缓存的所有页(索引和数据)。理想情况下，该比率应当超过 96%，并尽可能接近 100%。要提高缓冲池命中率，请尝试下面这些方法：

- 增加缓冲池的大小。
- 考虑分配多个缓冲池。如果可能的话，为每个经常访问的大表所属的表空间分配缓冲池，为一组小表分配缓冲池，然后尝试使用不同大小的缓冲池查看哪种组合会提供最佳性能。

如果已分配的内存不能帮助提高性能，那么请避免给缓冲池分配过多的内存。应当根据取自测试环境的快照信息来决定缓冲池的大小。

对于缓冲池的管理还应该遵循如下大的原则：对于 OLTP 系统，一般需要为不同类型的数据创建不同的缓冲池来提高某些频繁访问数据的访问效率；而对于 DSS/OLAP 系统，

一般只创建很少数量或只有一个缓冲池，但这些缓冲池的大小应该足够大，以满足对大量数据进行处理的需要。

5. pckcachesz

定义：

此参数是在数据库共享内存之外分配的，并且用于高速缓存数据库上的静态和动态 SQL，以及 XQuery 语句的所有 section。

默认值：

DB2 V9.5　32 位 automatic [-1, 32-128000]；64 位 automatic [-1, 32-524288]

DB2 V9.7　32 位 automatic [-1, 32-128000]；64 位 automatic [-1, 32-2147483646]

计量单位：页 (4 KB)

是否可以联机配置：

是

设置指引：

该参数在 OLTP 环境中比较重要，程序包高速缓存(pckcachesz)用来缓存数据库上的 SQL 部分程序包，使数据库管理器在重新装入程序包时可以不访问系统目录；或者对于动态 SQL 或 XQuery 语句，可以免去编译这一步，从而减少内部开销；对于在与数据库连接的应用程序多次使用同一条语句时，可以大大提高性能。

所以，该值一般不建议设置为 automatic，应根据实际需求设置大小，建议设置为 256MB(65536*4KB)或更大值。

```
    db2 update db cfg using pckcachesz 65536
```

那么，数据库运行一段时间后，怎么判断这个参数的设置是否满足要求呢？

```
    select (1-PKG CACHE INSERTS/PKG CACHE LOOKUPS)*100 as PCHR,
PKG_CACHE_NUM_OVERFLOWS from sysibmadm.snapdb
```

例如：

```
    $db2 "select (1-PKG CACHE INSERTS/PKG CACHE LOOKUPS)*100 as PCHR,
PKG CACHE NUM OVERFLOWS from sysibmadm.snapdb "

    PCHR         PKG CACHE NUM OVERFLOWS
    --------------------  -----------------------
     100           0
      1 record(s) selected.
```

如果 PCHR<100%或 PKG_CACHE_NUM_OVERFLOWS>0，就需要考虑适当增大 pckcachesz 的大小。

6. catalogcache_sz

定义：
此参数指定目录高速缓存可以使用的数据库堆中的最大空间(以页计)。

默认值：
-1 [MAXAPPLS*5]

是否可以联机配置：
是

设置指引：

此参数是在数据库共享内存之外分配的，并且用于高速缓存系统目录信息。在分区数据库系统中，每个数据库分区都有一个目录高速缓存。

高速缓存各个数据库分区中的目录信息允许数据库管理器不需要访问系统目录(或分区数据库环境中的目录节点)就可以获取先前检测的信息，从而降低内部开销。使用目录高速缓存(catalogcache-sz)可以帮助提高下列各项的整体性能：

- 绑定程序包以及编译 SQL 和 XQuery 语句
- 涉及检查数据库级别特权、例程特权、全局变量特权和角色权限的操作
- 连接至分区数据库环境中非目录节点的应用程序

建议刚开始使用默认值，并使用数据库系统监视器来进行调整。当调整此参数时，应考虑如果为目录高速缓存保留的额外内存是为另一目的分配的(如缓冲池或程序包高速缓存)，值是否会更有效。如果工作负载涉及在短时间内编译许多 SQL 或 XQuery，并且随后很少或不进行编译，那么调整此参数尤其重要。如果高速缓存太大，那么可能会因保留不再使用的信息的副本而浪费内存。在分区数据库环境中，考虑是否需要将目录节点上的 catalogcache_sz 设置为更大的值，因为非目录节点上要求的目录信息将始终首先在目录节点上高速缓存。

如何监控目录缓存的使用情况呢？有两种方法：

通过数据库系统监视器：

```
$db2 get snapshot for database on npsm |grep -i "catalog"
Catalog database partition number          = 0
```

```
    Catalog network node name                    =
    Catalog cache lookups                       = 178777724
    Catalog cache inserts                       = 2280
    Catalog cache overflows                      = 0
    Catalog cache high water mark                 = 1251679
    Catalog cache statistics size                 = 0
        Memory Pool Type                           = Catalog Cache Heap
```

通过 SQL 语句：

```
    $db2 "select (1-CAT CACHE INSERTS/CAT CACHE LOOKUPS)*100 as CCHR,
CAT CACHE OVERFLOWS from sysibmadm.snapdb"

    CCHR        CAT CACHE OVERFLOWS
    -------------------- --------------------
     100          0

      1 record(s) selected.
```

如果命中率(1－(Catalog cache inserts/Catalog cache lookups)) * 100<95%，可以考虑增加该参数的值。

如果 Catalog cache overflows 不为 0，也需要增加该参数的值，一般同时会增加 dbheap 参数的值。

调整办法：

```
    db2 update db cfg using CATALOGCACHE_SZ XXXX
```

10.4　锁有关的设置

1. DB2_EVALUNCOMMITTED

定义：

当启用此变量时，可对未落实的数据进行谓词求值。

DB2 V8.1.4 中首次引入了 DB2_EVALUNCOMMITTED 这个 DB2 注册变量。

DB2_EVALUNCOMMITTED 变量影响 DB2 在游标稳定性(CS)和读稳定性(RS)隔离级别下的行锁机制。当启用该功能时，DB2 可以对未提交的插入(INSERT)或更新(UPDATE)数据进行谓词判断，如果未提交数据不符合该条语句的谓词判断条件，DB2 将不对未提交数据加锁，这样就避免了因为要对未提交数据加锁而引起的锁等待状态，提高了应用程序访问的并发性。同时 DB2 会无条件忽略在进行表扫描时删除的行数据(不管是否提交)。详

细的解释和测试结果可以参考《高级进阶 DB2(第 2 版)》中的锁和并发相关章节。

默认值：

NO

是否可以联机配置：

否

设置指引：

这个参数建议生产数据库上配置成 ON，需要重启实例才能生效。

从 DB2 命令窗口输入 db2set DB2_EVALUNCOMMITTED 命令：

```
db2set DB2 EVALUNCOMMITTED=ON
db2stop force
db2start
```

2. DB2_SKIPINSERTED

定义：

DB2_SKIPINSERTED 注册变量控制对于使用游标稳定性(CS)或读稳定性(RS)隔离级别的语句，是否可以忽略未落实的数据插入。

默认值：

OFF

是否可以联机配置：

否

设置指引：

根据 DB2_SKIPINSERTED 注册变量的值不同，将以两种方式中的一种来处理未落实的插入。

如果值为 ON，DB2 将把未提交的 INSERT(只对于 CS 和 RS 隔离级别)看做它们还没有被插入，这在许多情况下能够提高并行性，并且是大多数应用程序的首选行为。未落实的插入被视为尚未发生。

如果值为 OFF(默认值)，DB2 服务器将等待插入操作完成(落实或回滚)，然后相应地处理数据。详细的说明和测试结果见《高级进阶 DB2(第 2 版)》中的锁和并发章节。

建议在生产数据库上配置成 ON。从 DB2 命令窗口输入 db2set DB2_SKIPINSERTED 命令：

```
db2set DB2_SKIPINSERTED=ON
db2stop force
db2start
```

3. cur_commit

定义：

cur_commit(当前已落实)是 DB2 V9.7 中引入的 DB CFG 参数，在先前版本中，CS 隔离级别会阻止应用程序读取其他应用程序更改过的任何行，直到更改已落实。在 DB2 V9.7 中，在 CS 隔离级别下，启用当前已落实的语义之后，读操作不必等待行更改落实即可返回值，尽可能返回当前已落实的结果，大大提高了并发性。

默认值：

ON

是否可以联机配置：

否

设置指引：

建议保持默认设置就可以。需要提醒的一点是，如果 DB2 数据库是从 9.5 升级到 9.7 版本，在数据库升级期间，cur_commit 配置参数将设置为 DISABLED，需要升级后手工更改成 ON。

在启用此参数后，相对于未启动此参数的数据库有两个方面的资源需要相应增加：日志空间、日志缓冲(logbufsz)和缓冲池(bufferpool)。在使用 cur_commit 的设置后，对这两方面的资源消耗会相应增加，所以应适当增大设置的值。

4. locktimeout

定义：

此参数指定应用程序为获取锁定将等待的秒数，以帮助避免应用程序出现全局死锁。

默认值：

-1 [-1；0 - 32767]

是否可以联机配置：

否

设置指引：

如果锁定请求处于暂挂的时间大于 locktimeout 值，那么请求应用程序将接收到错误并将其事务回滚。locktimeout 的默认值为－1，可以关闭锁定超时检测，如果出现锁等待，应用程序将会出现无穷等待现象。例如，如果 APPL1 尝试获取已由 APPL2 挂起的锁定，那么 APPL1 在超时周期到期时返回 SQLCODE－911 (SQLSTATE 40001)，原因码为 68。对于生产系统中的 OLAP，locktimeout 一开始为 60(秒)比较好，对于 OLTP 大约为 10 秒比较好。对于开发环境，应该使用－1 以识别和解决锁等待的情况。如果有大量的并发用户，可能需要增加 locktimeout 时间以避免回滚。

在事务处理(OLTP)环境中，可以使用 30 秒的初始启动值。在只查询的环境中，可以从一个较高的值开始。

```
db2 update db cfg using locktimeout 30
db2stop force
db2start
```

5. locklist

定义：

此参数指示分配给锁定列表的内存量。每个数据库都有一个锁定列表，锁定列表包含由同时连接至数据库的所有应用程序挂起的锁定。

默认值：

automatic [4 - 524288]

是否可以联机配置：

是

设置指引：

应用程序使用的锁定列表百分比达到 maxlocks 时，数据库管理器就会对应用程序挂起的锁定执行从行到表的锁定升级。虽然升级过程本身花不了多少时间，但是锁定整个表(与个别行比较)降低了并行性，并且可能因对受影响的表进行后续访问而降低整个数据库性能。如果锁定升级导致性能问题，那么可能需要增大此参数或 maxlocks 参数的值。可以使用数据库系统监视器来确定是否正在发生锁定升级。

```
db2 -v get snapshot for database on sample | grep -i lock
```

在快照输出中，检查下列各项：

```
Locks held currently = 21224
Lock waits = 24683
Time database waited on locks (ms) = 32875
Lock list memory in use (Bytes) = 87224
Deadlocks detected = 89
Lock escalations = 8
Exclusive lock escalations = 12
Agents currently waiting on locks = 0
Lock Timeouts = 0
Internal rollbacks due to deadlock = 0
```

如果“Lock list memory in use (Bytes)”超过定义的 locklist 大小的 50%，那么就增加 locklist 数据库配置参数中 4KB 页的数量。

如果发生了“Lock escalations>0”或“Exclusive lock escalations>0”，就应该增大 locklist 或 maxlocks，抑或同时增大两者。

建议使用默认值 automatic，并根据实际情况调整大小。

```
db2 update db cfg using locklist XXXXX
```

6. maxlocks

定义：

此参数定义应用程序挂起的锁定列表的百分比，必须在数据库管理器执行锁定升级之前填写该列表。

默认值：

automatic [1 - 100]

是否可以联机配置：

是

设置指引：

maxlocks 是与 locklist 参数一起调整的，建议使用默认值。

建议使用默认值 automatic，并根据实际情况调整大小。

```
db2 update db cfg using maxlocks  XXXXX
```

10.5 日志相关的配置

1. DB2_LOGGER_NON_BUFFERED_IO

定义：
此变量将在日志文件系统上启用直接 I/O。

默认值：
DB2 V9.5 默认是关闭的(OFF)。
DB2 V9.7 默认开启，并且在默认情况下主日志文件和辅助日志文件使用非缓冲 I/O。

是否可以联机配置：
否

设置指引：
建议针对 DB2 V9.5 将该参数设置为 ON。
从 DB2 命令窗口输入 db2set DB2_LOGGER_NON_BUFFERED_IO 命令：

```
db2set DB2_LOGGER_NON_BUFFERED_IO =ON
```

2. logarchmeth1

定义：
此参数指定已归档日志的主要目标的介质类型。

默认值：
OFF [LOGRETAIN, USEREXIT, DISK, TSM, VENDOR]

是否可以联机配置：
否

设置指引：
交易系统必须启用归档设置，但可以根据情况不同选择设置为目录或带库。建议配置到本地磁盘，再传到 NBU 带库。该参数启用之后，必须对数据库做离线全备份。

```
db2 update db cfg using  LOGARCHMETH1 disk:/dir  =>设置成磁盘目录
或者
db2 update db cfg using LOGARCHMETH1 VENDOR:/usr/openv/netbackup/bin/nbdb2.sl64
```

3. logbufsz

定义：

在将日志记录写入磁盘之前，此参数允许您指定用作这些记录的缓冲区的数据库堆大小(由 dbheap 参数定义)。

默认值：

DB2 V9.5　32 位　8 [4 - 4096]；64 位　8 [4 - 131070]

DB2 V9.7　32 位　8 [4 - 4096]；64 位　256 [4 - 131070]

计量单位：

页(4KB)

是否可以联机配置：

否

设置指引：

logbufsz 是数据库配置参数，用于日志缓冲区。这个参数允许指定用作在将日志记录写到磁盘之前的缓冲区的数据库堆(dbheap)的数量。当提交事务或者日志缓冲区已满的时候，就要将日志记录写入磁盘。对日志记录进行缓冲将导致将日志记录写入磁盘的活动不那么频繁，但每次要写的日志记录会更多。这个多数允许指定数据库共享内存的大小以用作在将日志记录写到磁盘之前这些记录的缓冲区。当下列事件之一发生时会将日志记录写到磁盘：

- 事务提交。
- 日志缓冲区已满或 1 秒钟。
- 其他某个内部数据库管理器事件发生。

将日志记录存放在缓冲区将产生更加有效的日志文件 I/O，这是因为这样一来可以降低将日志记录写到磁盘的 I/O 频率，同时每次可写更多的日志记录。如果对专用的日志磁盘有相当多的读操作，或者希望有较高的磁盘利用率，那么可以增加这个缓冲区的大小。当增加这个参数的值时，也要考虑 dbheap 参数，因为日志缓冲区使用的空间由 dbheap 参数控制。

通过查看下面这个示例中的各行，使用数据库快照来确定 logbufsz 参数的值是否为最佳：

```
Log pages read = 0
Log pages written = 12644
```

对于 OLTP，一开始至少设置为 256 页；对于 OLAP，则以 128 页为佳。如果常常看到大于 1 的"Log pages read"值，那么可能需要增加这个值。如果发生回滚，也可能要读取日志数据。一般而言，“log pages read”和“log pages written”之比应当尽可能小。理想情况下，“log pages read”的值应为 0，而“log pages written”的值应很大。当“log pages read”太多时，意味着需要较大的 logbufsz。

如果在试图增大 logbufsz 时返回错误，那么可以按相同大小增加 dbheap，然后再次尝试。

运行数据库，建议设置成 512 或更大数值。

```
db2 update db cfg using logbufsz 512
```

4. logfilsiz

定义：

此参数定义每个主日志文件和辅助日志文件的大小。在这些日志文件已满且需要新日志文件之前，这些日志文件的大小限制可写入这些日志文件的日志记录数。

默认值：

1000 [4 - 1048572]

计量单位：

页(4KB)

是否可以联机配置：

否

设置指引：

必须使日志文件的大小与主日志文件数平衡。

如果数据库要运行大量更新、删除或插入事务，而这将导致日志文件很快变满，那么应增大 logfilsiz 的值。日志文件大小的上限与日志文件数的上限(logprimary+logsecond)共同确定活动日志空间的上限。日志文件太小则会因归档旧日志文件、分配新日志文件以及等待可用日志文件的开销而影响系统性能。

如果磁盘空间不足，那么应减小 logfilsiz 的值，因为主日志是按此大小预分配的。太大的日志文件会减小管理归档日志文件和日志文件副本时的灵活性。

该参数的默认值应设置比较小，建议设置成 10240 或更大数值。

```
db2 update db cfg using logfilsiz 10240
```

5. logprimary

定义：

此参数允许您指定要预分配的主日志文件数。主日志文件建立分配给恢复日志文件的固定存储器数量。

默认值：

3 [2 - 256]

是否可以联机配置：

否

设置指引：

为此参数选择的值取决于许多因素，包括正在使用的记录类型、日志文件的大小和处理环境的类型(例如事务的长度和落实的频率)。增大此值将增大日志的磁盘要求，因为主日志文件就是在第一个与数据库的连接期间预分配的。如果您发现经常分配辅助日志文件，那么可通过增大日志文件大小(logfilsiz)或增加主日志文件的数目来提高系统性能。

因为该参数的默认值很小，所以根据具体数据库需求，必须修改日志的数量，比如较小的交易系统可以设置为 13，较大的交易系统设置成 23 或更大。

```
db2 update db cfg using logprimary 23
```

6. logsecond

定义：

此参数指定(仅需要时)创建并用于恢复日志文件的辅助日志文件的数目。

默认值：

2 [-1； 0 -254]

是否可以联机配置：

是

设置指引：

对于定期需要大量日志空间的数据库，使用辅助日志文件。例如，每月运行一次的应用程序需要的日志空间可能会超过由主日志文件提供的日志空间。由于辅助日志文件不需要永久的文件空间，因此辅助日志文件在这种情况下有优势。

在生产数据库运行过程中，应该随时监控日志空间的使用情况，避免日志空间使用率

过大。可以使用下面的 SQL 语句监控日志空间的使用情况：

```
    select int(total log used/1024/1024) as "Log Used
(Mb)",int(total log available/1024/1024) as "Log Space Free(Mb)",
int((float(total log used)/float(total log used+total log available))*100)
as "Pct Used",int(tot log used top/1024/1024) as "Max Log Used (Mb)",
int(sec log used top/1024/1024) as "Max Sec. Used (Mb)",int(sec logs allocated)
as "Secondaries" from sysibmadm.snapdb

    输出：
    Log Used (Mb) Log Space Free(Mb) Pct Used   Max Log Used (Mb) Max Sec. Used
    (Mb)  Secondaries
    --------- ------------------ ----------- ----------------- ---------------
     67          14515           0              780             0           0

      1 record(s) selected.
```

如果发现日志空间使用率比较大，比如上面 SQL 查询的输出“Pct Used”超过了 50%，就应该引起我们的关注，临时的紧急解决办法就是在线扩大 logsecond 值来满足业务需求。

```
    db2 update db cfg using logsecond XXXXX
```

在此提醒大家，logsecond 参数有一个设置值-1，此值表示 logsecond 可以是无限大，这对于解决极端场景下日志不足的问题是很有帮助的。在设置此参数为-1 后，需要重点关注的就是日志目录所在文件系统的空间是否够用的问题了。

7. newlogpath

定义：

此参数允许您指定最多 242 个字节的字符串，以更改存储日志文件的位置。

默认值：

NULL [任何有效路径或设备]

是否可以联机配置：

否

设置指引：

理想情况是，这些日志文件将位于没有大量 I/O 的物理磁盘上。例如，避免将日志与操作系统或大容量数据库放在同一磁盘上。这将提高日志记录活动的效率并使开销(例如等待 I/O)最小。

根据实际需求，建议将此参数设置为特定文件系统，该文件系统存放在较快的磁盘上，并且与数据文件系统区分开。

```
db2 update db cfg using newlogpath XXXXX
```

10.6　自动维护相关的配置

1. auto_maint

定义：

此参数是所有其他自动维护数据库配置参数(auto_db_backup、auto_tbl_maint、auto_runstats、auto_stats_prof、auto_stmt_stats、auto_prof_upd 和 auto_reorg)的父参数。当禁用此参数时，也就禁用了它的所有子参数。但是，数据库配置文件中记录的这些子参数的设置不会更改。当启用此父参数时，记录的子参数的值就会生效。因此，可以全局启用或禁用自动维护。

默认值：

ON [ON；OFF]

设置指引：

使用默认设置 ON。默认情况下的输出如下，在默认设置下数据库会做自动统计信息更新。

```
$db2 get db cfg |grep -i "auto "
 Auto deletion of recovery objects    (AUTO_DEL_REC_OBJ) = OFF
 Automatic maintenance                      (AUTO_MAINT) = ON
   Automatic database backup            (AUTO_DB_BACKUP) = OFF
   Automatic table maintenance          (AUTO_TBL_MAINT) = ON
     Automatic runstats                 (AUTO_RUNSTATS) = ON
       Automatic statement statistics   (AUTO_STMT_STATS) = ON
     Automatic statistics profiling     (AUTO_STATS_PROF) = OFF
       Automatic profile updates          (AUTO_PROF_UPD) = OFF
     Automatic reorganization                (AUTO_REORG) = OFF
```

通过上面的输出，我们可以看到关于自动维护的设置有不少，这里重点介绍几个日常运行数据库不可或缺的设置。

2. automatic runstats-auto_runstats

此设置默认为 ON，这个设置用于设置数据库是否启用自动的 RUNSTATS 工具进行统计信息更新。如果设置为 ON，DB2 就会根据一系列内置的规则来决定需要对哪些表和索引运行 RUNSTATS 工具。这一设置为默认启动机制，并且在绝大多数 DB2 数据库中都运行良好，因此一般建议数据库能够使用这个功能。

3. automatic statement statistics-auto_stmt_stats

此设置在 DB2 V9.7 下默认设置为 ON，此设置用于指定数据库是否启动实时(real-time)收集统计信息功能，启用这一功能后，在执行每条 SQL 的时候，DB2 会对 SQL 中涉及的表和索引进行评估，如果满足条件就触发同步的统计信息收集，收集完成后 SQL 才能继续执行。这种方式较同步信息收集方式，除了这种方式之外，还可以触发异步信息收集，也就是在收集统计信息的时候不会影响 SQL 的执行。

关于上面提到的两种自动化的统计信息收集功能的执行情况，我们可以通过 $INSTANCDIR/sqllib/db2dump/events 目录下的文件来查看，这些文件详细记录了这些统计信息收集的启动和结束时间及过程，或是失败的状态。

10.7 监控相关的配置

dft_monswitches

定义：

dft_mon_uow

快照监视器的工作单元(UOW)开关的默认值

dft_mon_stmt

快照监视器的语句开关的默认值

dft_mon_table

快照监视器的表开关的默认值

dft_mon_bufpool

快照监视器的缓冲池开关的默认值

dft_mon_lock

快照监视器的锁定开关的默认值

dft_mon_sort

快照监视器的排序开关的默认值

dft_mon_timestamp

快照监视器的时间戳记开关的默认值

默认值：

默认是所有开关关闭，但 dft_mon_timestamp 除外，该开关在默认情况下是打开的。

设置指引：

建议都打开，如果在修改 dft_mon_xxxx 开关设置前显式连接至实例，那么更改将立即生效。否则，设置在下次重新启动实例时生效。

```
    $db2 update dbm cfg using DFT_MON_BUFPOOL on DFT_MON_LOCK on DFT_MON_SORT
on DFT_MON_TABLE on DFT_MON_TIMESTAMP on DFT_MON_UOW on DFT_MON_STMT ON
```

在这里需要提醒大家注意的一点就是，对于监控开关的设置通常有两个级别：实例级别和会话级别。实例级别的设置在实例范围内全局生效；而会话级别的设置则只在当前的会话中生效，其他会话、甚至是后续新建立的会话都不会受此影响。

在数据库实例内部还存在监控开关的一种状态，在实例底层保存的这些状态表示实例本身为这些监控开关申请的资源是否存在，可以用下面的命令来查看：

```
    $db2 get snapshot for database manager
              Database Manager Snapshot

    Node type                                  = Enterprise Server Edition with
local and remote clients
    ..........................
    Switch list for db partition number 0
    Buffer Pool Activity Information  (BUFFERPOOL) = ON  01/21/2013 16:47:15.835304
    Lock Information                    (LOCK)  = ON  01/21/2013 16:47:15.835304
    Sorting Information                 (SORT)  = ON  01/21/2013 16:47:15.835304
    SQL Statement Information         (STATEMENT)  = ON  01/21/2013 16:47:15.835304
    Table Activity Information           (TABLE)  = ON  01/21/2013 16:47:15.835304
    Take Timestamp Information        (TIMESTAMP)  = ON  01/21/2013 16:47:15.835304
    Unit of Work Information               (UOW)  = ON  01/21/2013 16:47:15.835304
    ..........................
```

10.8 安全相关的设置

在实例中，和安全相关的配置参数有很多，这个通常和客户需要配置的安全环境有关。下面我们主要介绍对于安全比较重要的实例配置参数。

AUTHENTICATION

定义：
指定并确定如何进行以及在何处进行用户认证。

默认值：
SERVER

设置指引：
该参数控制实例的安全认证方式。默认为 SERVER，表示在服务器上验证用户和密码，这个参数我们在《高级进阶 DB2(第 2 版)》一书的安全部分有非常详细的讲解，读者可以参考。

建议设置成 SERVER_ENCRYPT，这意味着服务器接受已加密的 SERVER 认证方案并对用户数据进行加密。

```
$db2 update dbm cfg using AUTHENTICATION SERVER_ENCRYPT
```

在这里需要提醒大家的一点是：在 AIX 操作系统上，在 DB2 V9.7.6 及后续的版本中，由于操作系统的认证模块方面的缺陷，造成 CLI 方式连接数据库的时间很长，建立连接的时间可能达到 10 分钟以上，为了避免这一问题，可以将参数 AUTHENTICATION 设置为 SERVER，这样就会让 DB2 无须调用操作系统有缺陷的认证模块，从而避免这一问题。

此外，参数 SYSADM_GROUP、SYSCTRL_GROUP、SYSMAINT_GROUP 和 SYSMON_GROUP 决定我们如何为不同的组设置不同的安全管理权限。关于这几个参数和安全相关的概念，读者可以参考《高级进阶 DB2(第 2 版)》一书的安全部分。

10.9 供参考的 DB2 上线前设置

表 10-2 是笔者在实际生产数据库上线前，针对 DB2 上线配置中的一部分，供大家学习参考，以便了解 DB2 参数在实际生产中的应用和设置情况。

表 10-2　DB2 上线前的参数检查列表

分类	描　述	设 置 方 法	备　注
注册变量	设置 DB2_EVALUNCOMMITTED 为 ON	db2set　DB2_EVALUNCOMMITTED=ON	重启实例生效
	设置 DB2_SKIPINSERTED 为 ON	db2set　DB2_SKIPINSERTED=ON	重启实例生效
	DB2COMM 是否设置为 TCPIP	db2set　DB2COMM=TCPIP	重启实例生效
	DB2_PARALLEL_IO	db2set DB2_PARALLEL_IO=*	重启实例生效
	设置日志访问为 DIO	db2set DB2_LOGGER_NON_BUFFERED_IO=ON	重启实例生效
实例参数设置	开启所有实例监控开关	db2 update dbm cfg using DFT_MON_BUFPOOL on DFT_MON_LOCK on DFT_MON_SORT on DFT_MON_TABLE on DFT_MON_TIMESTAMP on DFT_MON_UOW on DFT_MON_STMT ON	在线生效
	关闭健康监控	db2 update dbm cfg using HEALTH_MON OFF	在线生效
	设置使用认证模式为 SERVER_ENCRYPT	db2 update dbm cfg using AUTHENTICATION SERVER_ENCRYPT	SERVER_ENCRYPT
	instance_memory	db2 update dbm cfg using instance_memory SIZE	非独立数据库服务器根据实际需求指定大小
	设置 svcename	db2 update dbm cfg using svcename DB2_instance	特定端口或对应的名称
	设置独立的 sysmon 组	db2 update dbm cfg using SYSMON_GROUP groupname	

(续表)

分类	描　述	设 置 方 法	备　注
数据库设置	建库的 CODEPAGE 为 UTF-8	db2 get db cfg|grep "code page"	1208
	database_memory	db2 update db cfg using database_memory SIZE	非独立数据库服务器根据实际需求指定大小
	必须修改数据库日志文件大小：logfilsiz	db2 update db cfg using logfilsiz 20480	10240 或更大
	修改日志数量，根据具体数据库需求，必须修改日志的数量，如较小的交易系统可以设置为 13 和 27	db2 update db cfg using logprimary 23 logsecond 37	需要重启数据库使其生效
	必须修改日志缓冲：logbufsz	db2 update db cfg using LOGBUFSZ 512	512 或更大
	交易系统必须启用归档设置，但可以根据情况不同选择设置为目录或带库	db2 update db cfg using LOGARCHMETH1 disk:/dir 或 db2 update db cfg using LOGARCHMETH1 VENDOR:/usr/openv/netbackup/bin/nbdb2.sl64	需要重启数据库使其生效
	修改 locktimeout	db2 update db cfg using locktimeout 30	需要重启数据库使其生效
	必须修改默认缓冲池 ibmdefaultbp 大小	db2 "alter bufferpool ibmdefaultbp size 32768" db2pd -db dbname -bufferpools	具体值根据需要确定
	pckcachesz	db2 update db cfg using pckcachesz SIZE	根据实际需求设置大小，256MB 或更大
	必须修改默认日志路径	db2 update db cfg using newlogpath /path	根据实际需求，设置为特定文件系统

10.10　本章小结

本章结合笔者的实际运维经验，重要介绍了运行数据库必须考虑的数据库参数。正如本章开头提到的那样，DB2 数据库中的参数有很多，像注册变量、实例参数、数据库参数、表空间参数、表参数等，不胜枚举，而且随着业务系统运行一段时间之后，业务行为和业务量都在不断发生变化，很多原先设置的参数可能都需要进行调整，才能达到性能最优。所以，本章只是起到一个入门介绍的作用，用来帮助读者了解 DB2 各个重要参数的意义和设置时的考虑因素。更深入的知识可以参考 DB2 信息中心来进一步学习。



DBA 日常运行维护

DB2 数据库在日常运行过程中，DBA 需要经常做一些维护性的工作来确保数据库稳定、高效运转。这些维护性工作主要包括统计信息更新(RUNSTATS)、碎片整理(REORG)、程序包重新绑定(REBIND)、健康检查、数据库监控等。这些日常维护工作对于数据库来说至关重要，本章主要讲解如下内容：

- 统计信息更新
- 统计信息更新案例分析
- 碎片整理
- 碎片整理案例分析
- 程序包重新绑定
- 健康检查
- 数据库监控

11.1 统计信息更新

统计信息是 DB2 收集的关于数据库中各个数据对象状态的信息，这些信息在收集好以后被保存在数据库系统编目表中，当应用程序或 SQL 语句对数据进行访问的时候，优化器需要根据这些统计信息来生成成本最低的执行计划。只有准确的统计信息才能让 DB2 优化器产生最优的执行计划，进而提高数据访问效率。如果数据库中的统计信息是过时的或者没有相关的统计信息，那么数据的高效访问就无从谈起。所以统计信息的准确与否就显得非常重要。

11.1.1 统计信息的重要性

DB2 优化器是 DB2 的“大脑”，而 DB2 优化器正是依靠存放在系统目录表中的数据库统计信息才能做出正确的判断，估算出某个特定 SQL 语句的所有执行计划的成本，随后 DB2 优化器会选择成本最低的执行计划来获取数据。读者可能会问，成本究竟表示什么呢？成本是由 CPU 成本(以指令数计)和 I/O(以寻道数和页的转换数计)的组合得出的。成本的单位是 timeron。timeron 不直接等于任何实际的所用时间，只是给出粗略估计的资源(成本)，估计的资源是数据库管理器执行同一查询的两种方案所必需的。在这里需要注意，最常见的查询操作是 select 语句，而 DML(比如 update)或 DDL(比如索引重建)也会执行查询操作，这时一样需要通过基于成本的 DB2 优化器进行分析。

数据统计信息中的变化会影响 DB2 优化器对获取目标数据的访问路径成本的估算，从而可能选择不同的访问路径。因此，如果数据库统计信息误差过大，就有可能造成性能问题。

例如，一张有 100 万行数据的表，数据库统计信息却记录这个表只有一行数据，那么 DB2 根据统计信息就有可能选择全表扫描而不是索引去获取数据。

下面列举了一些统计信息，这些统计信息可以帮助优化器选择最优访问策略：

- 表中的页数和非空的页数。
- 表中发生行链接的数量。
- 表中的行数。
- 有关单个列的统计信息，比如一列中唯一值的数量。
- 索引的聚合程度，也就是表中数据的存储顺序与某索引字段顺序的符合程度。
- 有关索引的统计信息，比如索引级别的数量和每个索引中叶子页的数量。
- 经常使用的列值的出现次数。
- 列值在列中所有值中的分布状况。
- 用户定制函数(UDF)的成本估计。

除以上信息外，DB2 还可以收集下列信息：索引的聚合程度、索引中的叶子页数目、溢出原始页的表行数以及表中已填充的页数和空页数。DBA 可以参考这些信息来决定何时重组表和索引。

DB2 不会在每次数据库添加、删除、更新数据后都实时更新数据库统计信息，这是因为过于频繁地更新统计信息有可能造成系统性能的巨大开销；此外，如果改变的数据量有限，那么有可能统计信息的一些微误差不会造成 DB2 选择成本高昂的访问路径，这时也没有必要频繁更新数据库统计信息。

更新统计信息有两种方式：自动和手动。打开数据库参数 AUTO_RUNSTATS 即可开

启自动统计信息收集方式，每两个小时触发一次。信息收集的规则如图 11-1 所示。

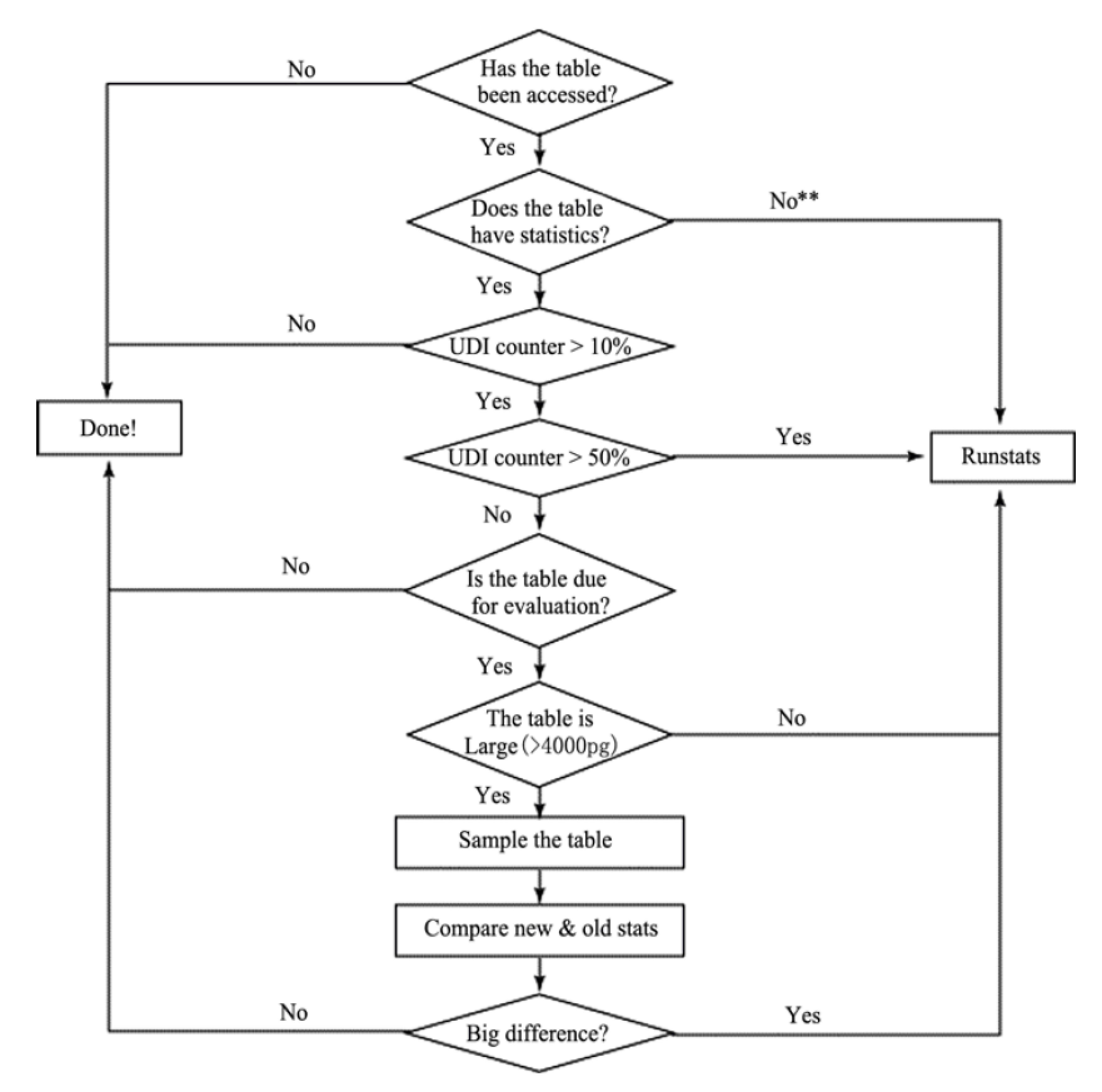


图 11-1　信息收集的规则

有些时候，自动收集方式并不是很及时，或者在收集统计信息的时候需要指定特殊的参数，这时就需要用 DB2 提供的 RUNSTATS 命令来手工更新数据库统计信息。当用户对表做了大量的数据更新后，可以考虑在表和索引上执行 RUNSTATS 命令以确保系统目录表中的统计信息是最新的。

在成功执行 RUNSTATS 命令之后，静态 SQL 查询并不会使用最新的数据库统计信息，这是因为静态 SQL 的访问策略在之前执行 BIND 时就已确定，而当时使用的统计信息有可能与现在的并不一致。这时就需要重新绑定使用静态 SQL 的应用程序，这样查询优化器就可以根据数据库的最新统计信息选择获取数据的最佳访问策略。但是，对于使用动态 SQL

的应用程序而言，没必要进行重新绑定，因为动态 SQL 语句的访问策略是根据统计信息在运行时动态生成的。

现在，几乎所有主流数据库都使用某种方法(在 Oracle 数据库中调用 dbms_stats 存储过程；在 Informix 数据库中用 update statistics；在 Sybase 数据库用 update statistics)来更新系统目录统计信息，以便为优化器提供可能的最佳信息。可以将优化器想象成汽车的 GPS 定位仪，能够在由系统数据组成的莽莽深山里做行驶路径选择。目录统计信息的更新将为优化器提供最新、最准确的地图信息，以便能够获取最佳行驶路径。

1. 如何更新统计信息

只有当进行显式请求时，对象的统计信息才会在系统目录表中被更新。有以下几种方法可以更新部分或全部统计信息：

- 使用 RUNSTATS(运行统计信息，run statistics)命令。
- 使用带有指定的统计信息收集选项的 LOAD。
- 对针对一组预先定义的系统目录视图进行操作的 SQL UPDATE 语句进行编码。
- 使用 reorgchk update statistics 命令。

当你不完全知道所有表名或表名实在太多而无法对每张表逐个更新策略时，进行 RUNSTATS 的最简单方法就是使用 db2 reorgchk update statistics on table all 命令，输出结果如下：

```
Doing RUNSTATS ....


Table statistics:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (Effective Space Utilization of Data Pages) > 70
F3: 100 * (Required Pages / Total Pages) > 80

SCHEMA.NAME                    CARD     OV     NP     FP ACTBLK    TSIZE  F1
F2  F3 REORG
----------------------------------------------------------------------------
Table: DB2TEST1.ACT
                                0      0      0      1      -        0   0    -    0 ---
Table: DB2TEST1.ADEFUSR
                                0      0      0      1      -        0   0    -    0 ---
Table: DB2TEST1.CC
                                1      0      1      2      -       15   0    0  100
---
```

```
Table: DB2TEST1.CC_EXCEPTION
                             1      0      1      1      -       15    0    - 100
---
Table: DB2TEST1.CL_SCHED
                             0      0      0      1      -        0    0    -    0 ---
Table: DB2TEST1.DB1208
                             3      0      1      1      -       45    0    - 100
---
……

Index statistics:

F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
F5: 100 * (Space used on leaf pages / Space available on non-empty leaf pages)
> MIN(50, (100 - PCTFREE))
F6: (100 - PCTFREE) * (Amount of space available in an index with one
less level / Amount of space required for all keys) < 100
F7: 100 * (Number of pseudo-deleted RIDs / Total number of RIDs) < 20
F8: 100 * (Number of pseudo-empty leaf pages / Total number of leaf pages)
< 20

SCHEMA.NAME              INDCARD  LEAF ELEAF LVLS  NDEL    KEYS LEAF RECSIZE
NLEAF RECSIZE LEAF PAGE OVERHEAD NLEAF PAGE OVERHEAD
PCT PAGES SAVED  F4  F5  F6  F7  F8 REORG
---------------------------------------------------------------------------
----------------------------------------------------
Table: DB2TEST1.ACT
Index: DB2TEST1.PK_ACT
                            0    1     0    1     0      0          2
2             822              822              0  100   -    -    0   0 -----
Index: DB2TEST1.XACT2
                            0    1     0    1     0      0          8
8             566              566              0  100   -    -    0   0 -----
```

我们上面所选的示例不需要表名。这一命令对所有表执行 RUNSTATS 命令。

记住：在批量数据加载后要运行 RUNSTATS 命令。

如果知道表名并且想避免对大量表执行 RUNSTATS 命令(因为这样做可能要花很长时间)，那么一次对一张表进行 RUNSTATS 更为可取。命令如下：

```
db2 -v runstats on table  tabschema.tabname and indexes all
```

这个命令将收集该表及其所有索引(基本级别)的统计信息。

2. 查看是否执行了 RUNSTATS 命令

要查看是否对某个表或索引执行了 RUNSTATS 命令，一种快捷方法便是查询系统目录表。

可以查询系统目录表中的以下列，确定是否在表和索引上执行了 RUNSTATS 命令：

- 如果对于某个表，SYSCAT.TABLES 视图的 STATS_TIME 列显示的值为 NULL，那么这表示没有对该表执行 RUNSTATS 命令。
- 如果对于某个索引，SYSCAT.INDEXES 视图的 STATS_TIME 列显示的值为 NULL，那么这表示还没有对该索引执行 RUNSTATS 命令。

```
表检查
db2 -v "select tbname, nleaf, nlevels, stats time from syscat.indexes"

索引检查
db2 -v "select tbname, nleaf, nlevels, stats_time from syscat.tables"
```

如果你认为 stats_time 列中显示的时间距离现在过久，那就需要再次执行 RUNSTATS 命令。

在表上执行 RUNSTATS 命令时，可以有两种用户访问选项：允许读访问 ALLOW READ ACCESS 和允许写访问 ALLOW WRITE ACCESS。在 执行 RUNSTATS 命令时如果加上 ALLOW READ ACCESS 选项，这时候其他用户只能够以只读的方式访问该表，这个选项会影响应用的并行性，因为任何想要更改表的操作这时都会处于等待状态。可以选择在使用 ALLOW READ ACCESS 选项时，同时使用 tablesample 选项，加上这个选项将只收集表的部分采样数据而不是所有数据。如果能合理选择采样数据大小，那么就可以在确保统计信息一致性的情况下，加快 runstats 的速度。

在执行 RUNSTATS 命令时如果加上 ALLOW WRITE ACCESS 选项，这时候其他用户可以读取或写入该表。在默认情况下，RUNSTATS 命令使用的是 ALLOW WRITE ACCESS 选项。

3. 分区数据库中的 RUNSTATS

在分区环境中，只需要在其中一个分区上发出 RUNSTATS 就可以对所有分区上的数据进行统计信息更新。

在 DB2 V9.5 之前，当在分区数据库上执行 RUNSTATS 命令，并且表分区位于发出 RUNSTATS 的数据库分区中时，RUNSTATS 将在该数据库分区上执行。如果表分区不在该数据库分区上，那么将请求发送给数据库分区组中持有该表分区的第一个数据库分区。

然后，在该数据库分区上执行 RUNSTATS 命令。

RUNSTATS 对分区收集统计信息时，有如下隐式假设：每个表中的行均匀分布在每个多分区数据库分区组中的所有分区上。

在下列情况下，使用 RUNSTATS 命令收集统计信息：

- 当向表装入数据并创建了新的索引时。
- 当用 REORG 命令重新组织表和索引时。
- 当存在大量影响表及其索引的更新、删除和插入操作时(此处的“大量”可能意味着 10%~20%的表和索引数据都受到了影响)。
- 在绑定对性能要求很高的应用程序之前。
- 当希望对新的和以前的统计信息进行比较时。定期进行统计使得能够在早期阶段发现性能问题。
- 当预存取(prefetch size)大小发生变化时。
- 当在表中创建新的索引时。如果自从上次在表中运行 RUNSTATS 以来尚未修改表，那么只需要对新的索引执行 RUNSTATS。
- 当想要比较当前和先前的统计信息时。如果定期更新统计信息，那么可以及早发现性能问题。
- 使用 RUNSTATS 命令收集关于 XML 列的统计信息。如果仅使用 RUNSTATS 收集 XML 列的统计信息，将保留 LOAD 或上一次执行 RUNSTATS 命令已收集的非 XML 列的现有统计信息。如果先前已收集关于一些 XML 列的统计信息，那么在当前命令未收集关于 XML 列的统计信息时，将删除先前收集的 XML 列的统计信息；在当前命令收集了关于 XML 列的统计信息时，将替换先前收集的 XML 列的统计信息。

要提高 RUNSTATS 性能并节省用来存储统计信息的磁盘空间，可以考虑仅指定需要收集数据分布统计信息的列。

如果没有足够的时间一次收集全部的统计信息，那么可以运行 RUNSTATS 来每次仅更新几个表、索引或统计信息视图的统计信息，并轮流完成该组对象。如果对选择性部分更新运行 RUNSTATS 期间，由于表上的活动而产生了不一致性，那么在查询优化期间将发出警告消息(SQL0437W，原因码 6)。例如，如果执行 RUNSTATS 来收集表分布统计信息，以及在某个表活动后，再次执行 RUNSTATS 来收集该表的索引统计信息，那么可能发生这种情况。如果由于表上的活动产生了不一致并且在查询优化期间检测到这些不一致，那么发出该警告消息。当发生这种情况时，应再次运行 RUNSTATS 来更新分布统计信息。

要确保索引统计信息和表同步，执行 RUNSTATS 来同时收集表和索引统计信息。索引

统计信息保留自上次运行 RUNSTATS 以来收集的大部分表和列的统计信息。如果自上次收集该表的统计信息以来已对该表做了大量修改，那么只收集该表的索引统计信息将使两组统计信息不能在所有节点上都同步。

在生产系统中调用 RUNSTATS 可能会对生产工作负载的性能产生负面影响。RUNSTATS 命令现在支持优先级选项，在执行较高级别的数据库活动期间，可以使用优先级选项来限制执行 RUNSTATS 的性能影响。

收集视图的统计信息时，将收集所有包含该视图引用的基本表的数据库统计信息。

可以考虑采用以下技巧来提高 RUNSTATS 的效率和已收集的统计信息的准确性：

- 仅对用来连接表的列或 WHERE、GROUP BY 以及查询的类似子句中的列收集统计信息。如果对这些列建立了索引，那么可以用 RUNSTATS 命令的 ONLY ON KEY COLUMNS 子句指定列。
- 为特定表和表中特定列定制 num_freqvalues 和 num_quantiles 的值。num_freqvalues 提供了重复最多的列和数据值的信息。num_quantiles 提供了数据值对于其他值而言是如何分布的有关信息。
- 使用 SAMPLED DETAILED 子句通过抽样计算详细的索引统计信息，这样就可以减少为获得详细索引统计信息而执行的后台计算量，使用 SAMPLED DETAILED 子句可以减少收集统计信息所需要的时间，并在大多数情况下产生足够的精度。
- 当创建已装载数据的表的索引时，添加 COLLECT STATISTICS 子句来在创建索引时创建统计信息，这一技巧在 Oracle 环境下也同样适用。
- 当添加或删除大量数据，或者更新收集了统计信息的列中的数据时，需要再次执行 RUNSTATS 命令来更新统计信息。
- 在 DB2 V9.5 之前，因为 RUNSTATS 仅收集单个数据库分区的统计信息，所以如果数据不是在所有数据库分区中一致分发的，那么统计信息将不太准确。如果怀疑存在不一致的数据分发，那么可能想要在执行 RUNSTATS 之前使用 REDISTRIBUTE DATABASE PARTITION GROUP 命令在各数据库分区之间重新分发数据。

可以通过比较查询 RUNSTATS 之前和之后的 SQL 语句的 EXPLAIN 输出，确定运行 RUNSTATS 对于访问计划的影响。

完成每一条 RUNSTATS 语句之后，都应该执行显式的 COMMIT 命令。COMMIT 将释放锁，并避免在收集多个表的统计信息时填写日志。

在用 RUNSTATS 收集了统计信息之后，使用 BIND 或 REBIND 命令重新绑定包含了静态 SQL 的应用程序包(并可以选择重新解释其语句)。db2rbind 命令可用于重新绑定数据库中的所有应用程序包。使用 FLUSH PACKAGE 命令删除程序包缓存器(package cache)中

当前所有缓存的动态 SQL 语句(db2 flush package cache dynamic)，并强制隐式地编译下一请求。

11.1.2　减小 RUNSTATS 对系统性能影响的策略

1. 调整 stat_heap_sz 数据库配置参数

统计信息堆(stat_heap_sz)这一数据库配置参数的大小指定了使用 RUNSTATS 命令收集统计信息中所用内存堆的最大值，是在启动 RUNSTATS 命令时分配的，然后在命令完成时释放。stat_heap_sz 是代理私有内存的一部分。因此，在收集统计信息时，最好增大 stat_heap_sz 参数，以便能将更多的统计信息放入这个堆中。显而易见，处理较大的表也需要更多的内存。当执行包含 SAMPLED DETAILED 选项的 RUNSTATS 命令时，将会额外多占用 2MB 的统计信息堆空间，因此必须分配更多内存，这样才能确保 RUNSTATS 命令能够更快完成。

2. 减小 RUNSTATS 对系统性能影响的策略

- 一次仅在少数表和索引上运行 RUNSTATS，在整组表中循环运行。
- 仅指定将收集数据分布统计信息的那些列。仅指定那些谓词中使用的列。
- 对于不同的表，在不同的分区上执行多个并发的 RUNSTATS 命令。
- 仅在那些迫切需要提高当前工作负载性能的关键表上执行 RUNSTATS 命令。避免在不需要它的表上运行 RUNSTATS 命令。
- 根据表中数据发生改变的速度，调整 RUNSTATS 命令定期执行的频度。
- 根据 RUNSTATS 命令在表上完成运行的速度，调整 RUNSTATS 的频率和采样数据的多少。
- 仅在系统活动量少的时候，安排执行 RUNSTATS。
- 调整(Throttle)RUNSTATS，以便最大程度地减少它对系统资源的需求。

仅在系统活动量少的时候安排执行 RUNSTATS 命令，这是最大程度地减少系统影响的一个好方法。然而，对于 24×7 的系统，系统中可能没有可用的时间窗口或活动量少的时候。处理该情形的一种方法就是使用 RUNSTATS 的 throttling 选项。

throttling 选项将根据当前的数据库活动级别，限制 RUNSTATS 命令占有的资源数量。在 DB2 中，在调整时，util_impact_lim 与 util_impact_priority 参数的配合使用确定了 RUNSTATS 命令的行为。util_impact_priority 关键字被用于 RUNSTATS 命令的子句中，而 util_impact_lim 则是实例配置参数。

util_impact_lim 参数是指允许所有运行的程序对于实例的工作负载产生影响的百分比。如果 util_impact_lim 是 100，就不用调整诸如 RUNSTATS 之类的运行程序所消耗的资源。例如，如果将 util_impact_lim 设置为 10(默认值)，那么正在运行的 RUNSTATS 命令就被限定消耗 10%以下的工作负载。

util_impact_priority 关键字可充当开关，指定 RUNSTATS 命令是否使用这种调整策略。

util_impact_priority 关键字的使用示例如下：

```
RUNSTATS ON TABLE db2admin.department AND INDEXES ALL UTIL_IMPACT_PRIORITY 10
```

其中，10 是调整 RUNSTATS 的优先权或级别。

当 util_impact_lim 不是 100，而且用 util_impact_priority 调用 RUNSTATS 时，将会对该值进行调整。如果没有为 util_impact_priority 指定优先权，并且 util_impact_lim 不是 100，那么 RUNSTATS 将使用默认的优先权 50，并将据此进行调整。如果在 RUNSTATS 命令中省略 util_impact_priority 关键字，那么不会对正在运行的 RUNSTATS 命令做资源调整。如果指定优先权，但是将 util_impact_limit 设置为 100，也可以不对正在运行的 RUNSTATS 命令做资源调整。如果将优先权设置为零，也可以不对正在运行的 RUNSTATS 命令做调整。

当定义了 RUNSTATS 调整(throttling)，并且该调整可操作时，RUNSTATS 命令通常会花费更长时间，但是对系统产生的影响会相对少一些。

11.1.3 DB2 自动统计信息收集

DB2 自动统计信息收集是在 DB2 UDB Version 8.2 中引入的。自动统计信息收集是完全自动进行表维护解决方案的一部分，其目标是允许 DB2 确定工作负载需要哪些统计信息，并定期在后台自动运行 RUNSTATS 命令，以便按需更新统计信息。

为了设置 SAMPLE 数据库自动进行统计信息收集，需要为自动维护开关设置数据库配置参数，如下所示：

```
db2 update db cfg for SAMPLE using AUTO MAINT ON
db2 update db cfg for SAMPLE using AUTO TBL MAINT ON
db2 update db cfg for SAMPLE using AUTO_RUNSTATS ON
```

图 11-2(只显示统计信息收集和配置选项)展示了统计信息收集和配置的自动维护命令的层次结构和相关性。在该结构中，可以在最高层次快速关闭自动维护参数 AUTO_ MAINT，而不丢失较低层的配置设置，比如 AUTO_RUNSTATS。

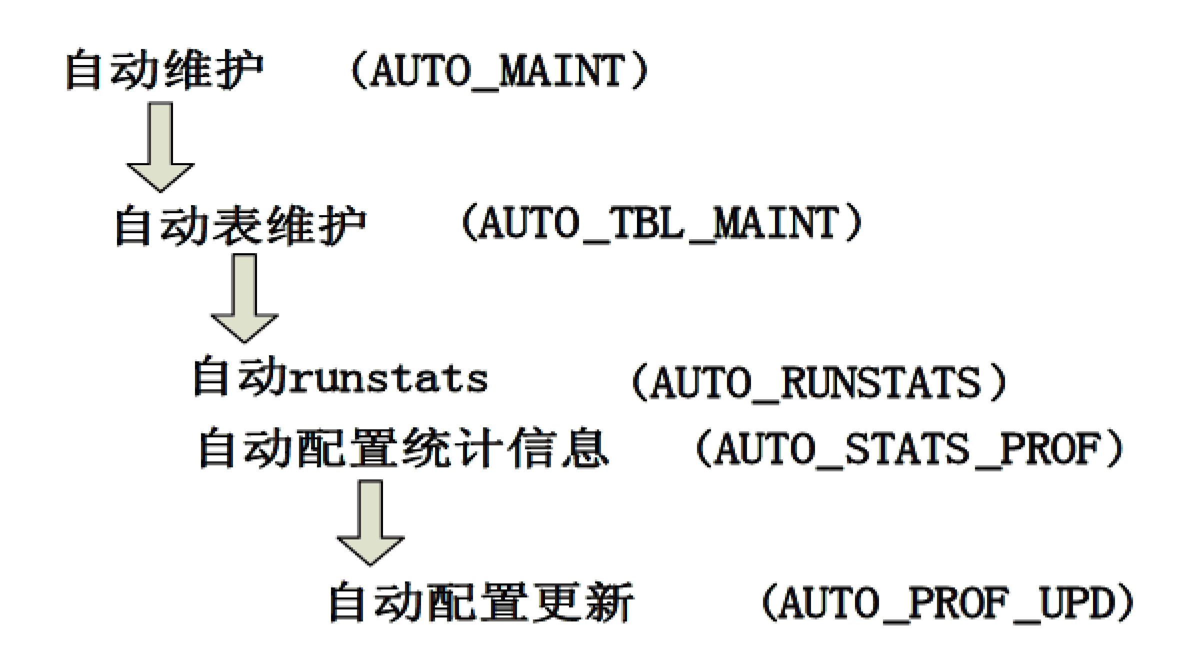


图 11-2　自动维护设置

如果需要自动化统计信息配置，那么可以打开参数 AUTO_STATS_PROF 和 AUTO_PROF_UPD。通过对自动化统计信息配置来确定何时和如何更进一步收集统计信息。统计信息配置文件是自动生成的，自动统计信息收集过程将用它来调度 RUNSTATS。可以用内部算法来比较新收集的统计信息与已保存的一组统计信息，并基于某些触发条件发出包含抽样的 RUNSTATS。

当启用自动统计信息配置时，数据库活动的有关信息被收集并存储在查询反馈库中。然后，基于查询反馈库中的数据生成统计配置文件。

为了允许自动生成统计信息配置文件，需要设置两个数据库配置参数：

- db2 update db cfg for SAMPLE using AUTO_STATS_PROF ON：打开该参数将启动查询反馈数据的收集。
- db2 update db cfg for SAMPLE using AUTO_PROF_UPD ON：打开该参数，指定使用分析查询反馈数据的 DB2 中的建议来更新 RUNSTATS 配置文件。

在触发自动统计信息配置之前，必须通过运行 SYSINSTALLOBJECTS 存储过程创建查询反馈库。

按照下列方式调用该存储过程：

```
call SYSINSTALLOBJECTS ( toolname , action , tablespacename , schemaname )
```

其中，toolname 是“ASP”或“AUTO STATS PROFILING”。对于 action，“C”表示创建，“D”表示删除。

例如，要创建反馈库，需要运行下列存储过程：

```
call SYSINSTALLOBJECTS ('ASP', 'C', 'USERSPACE1', '')
```

该存储过程将创建下列表、存储建议以及与查询执行过程中碰到的谓词有关的信息：

- SYSTOOLS.OPT_FEEDBACK_QUERY
- SYSTOOLS.OPT_FEEDBACK_PREDICATE
- SYSTOOLS.OPT_FEEDBACK_PREDICATE_COLUMN
- SYSTOOLS.OPT_FEEDBACK_RANKING
- SYSTOOLS.OPT_FEEDBACK_RANKING_COLUMN

当禁用 AUTO_PROF_UPD 参数时，可以将建议存储在 SYSTOOLS.OPT_FEEDBACK_RANKING 表中。然后，当手动更新 RUNSTATS 配置文件时，就可以查看在该表中存储的建议。

DB2 自动统计信息更新在实际运行中一般不使用，因为它是由数据库内部控制的，我们无法操纵，它很可能在你不期望出现的时候开始运行。在实际运行中，一般是根据自己的业务逻辑编写如下脚本，在合适的时间调度执行。

```
   #!/bin/ksh
# Source the db2 environment variables.
. @HOME@/sqllib/db2profile
LIST1=/tmp/tempreorg1
LIST2=/tmp/tempreorg2
LOG=@HOME@/schema/utilities/reorg.log
{
rm $LIST1 $LIST2 2>> $LOG
DATABASE=$1
#
db2 -v "connect to ${DATABASE}"
#
# get a list of tables
db2 "select tabname from syscat.tables where
tabschema='@SCHEMA@' and type='T'">$LIST1
if [ $? -ne 0 ]
then
print "an error occurred on connect"
exit 1
fi
#
# delete the first three lines
# delete the line that contains "record"
#
sed '1,3d' $LIST1 |sed '/record(s)/d' > $LIST2
#
#
print "begin reorg"
#
```

```
for i in `cat $LIST2`
{
# print reorg table $i
db2 -v "reorg table @SCHEMA@.$i"
if [ $? -ne 0 ]
then
print "An error occurred in reorg"
exit 1
fi
db2 -v "runstats on table @SCHEMA@.$i with distribution and detailed indexes
all"
if [ $? -ne 0 ]
then
print "an error occurred in runstats"
exit 1
fi
}
print "reorg SUCCESSFUL"
rm $LIST1 $LIST2
} > $LOG
```

11.2　统计信息更新案例分析

11.2.1　RUNSTATS 更新示例

下列举一些例子，说明如何使用 RUNSTATS 收集统计信息。

注意：

在RUNSTATS语法中，必须使用完全限定的表名schema.table-name和索引名schema.Index-name。可以在所有列上，或者仅仅在某些列或列组(除了LONG和LOB列)上执行RUNSTATS。如果没有指定特定列的子句，系统会使用默认的ON ALL COLUMNS子句。

收集所有列上的统计信息：

```
RUNSTATS ON TABLE db2admin.department ON ALL COLUMNS
```

这等同于：

```
RUNSTATS ON TABLE db2admin.department
```

收集单个列上的数据库统计信息：

```
RUNSTATS ON TABLE db2admin.department ON COLUMNS (deptno, deptname)
```

上面示例展示了如何收集关键列(key column)上的统计信息。短语“关键列”(key column)表示构成表上所定义索引的列。如果没有索引存在，这条命令不会收集任何列的统计信息。

收集关键列上的数据库统计信息：

```
    RUNSTATS ON TABLE db2admin.department ON KEY COLUMNS
```

收集关键列和非关键列上的数据库统计信息：

```
    RUNSTATS ON TABLE db2admin.department ON KEY COLUMNS AND COLUMNS (deptname)
```

收集表和索引上的数据库统计信息，不包含分布统计信息：

```
    RUNSTATS ON TABLE db2admin.department AND INDEXES ALL
```

收集表上的数据库统计信息以及索引上的详细统计信息，不包含分布统计信息：

```
    RUNSTATS ON TABLE db2admin.department AND DETAILED INDEXES ALL
```

只收集 3 个指定索引上的数据库统计信息(不收集表统计信息)：

```
    RUNSTATS ON TABLE db2admin.department FOR INDEXES db2admin.INX1,
db2admin.INX2, db2admin.INX3
```

只收集所有索引上的数据库统计信息：

```
    RUNSTATS ON TABLE db2admin.department FOR INDEXES ALL
```

如果创建新的索引，而在最后一次执行 RUNSTATS 以后还未修改相应的表，那么可以只收集索引上的数据库统计信息。在创建索引时，还可以收集数据库统计信息。

11.2.2 收集分布式统计信息

当确定表中数据分布不均匀时，可以运行包含 WITH DISTRIBUTION 子句的 RUNSTATS 命令。系统目录表中的统计信息通常包含关于表中最高值和最低值的信息，而优化器假定数据值是在两个端点值之间均匀分布的。然而，如果数据值彼此之间差异较大，或者群集在某些点上，或是碰到许多重复的数据值，那么优化器就无法选择最佳的访问路径，除非收集了分布统计信息。使用 WITH DISTRIBUTION 子句还可以帮助查询处理没有参数标记(parameter marker)或主机变量的谓词，因为优化器仍然不知道运行时的值是有许多行，还是只有少数行。

下面的例子说明了使用 RUNSTATS 收集包含数据分布信息的系统目录表统计信息的不同方法。

收集表和索引上的数据库统计信息，包含分布统计信息：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
```

收集表上的数据库统计信息以及索引上的详细统计信息，包含分布统计信息：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND DETAILED INDEXES ALL
```

收集选定列中包含分布的数据库统计信息：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS (deptno,
deptname)
```

只收集表上的数据库统计信息，包含 deptno 和 deptname 上的基本列统计信息以及 mgrno 和 admrdept 上的分布统计信息：

```
RUNSTATS ON TABLE db2admin.department ON COLUMNS (deptno, deptname)
  WITH DISTRIBUTION ON COLUMNS (mgrno, admrdept)
```

收集构成索引的所有列以及两个非索引列中包含分布的数据库统计信息：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON KEY
  COLUMNS AND COLUMNS (admrdept, location)
```

11.2.3　包含频率和分位数统计信息的 RUNSTATS

在执行包含 WITH DISTRIBUTION 子句的 RUNSTATS 时，会根据 RUNSTATS 命令中给定的选项选择一组频率(frequency)和分位数(quantile)的统计信息。

RUNSTATS 收集两种类型的数据分布统计信息：频率统计信息和分位数统计信息。

频率统计信息的默认值由 *num_freqvalues* 数据库配置参数控制，该值提供了重复最多的列和数据值的信息。默认值是 10，建议将这个值设置在 10~100 之间。如果将 *num_freqvalues* 设置为零，将不保留任何频率值的统计信息。

分位数统计信息的默认值由 *num_quantiles* 数据库配置参数控制，该值提供了数据值对于其他值而言是如何分布的有关信息。*num_quantiles* 数据库配置参数指定应将列数据值分成的组数。其默认值是 20，建议将该值设置在 20~50 之间。如果将这个参数设置为零或“1”，将不收集任何分位数统计信息。

如果没有在 RUNSTATS 命令的列或表级别上指定 num_freqvalues 和 num_quantiles，那么 num_freqvalues 的值将从 num_freqvalues 数据库配置参数中获取，而 num_quantiles 的值将从 num_quantiles 数据库配置参数中获取。

可以为单个列或一组列修改频率和分位数统计信息的精确度。提高分布统计信息的精

确度将导致更大的 CPU 和内存消耗，并占用更多的系统目录表空间。对于这些分布统计信息，只考虑对拥有选择谓词的最重要的查询而言最为重要的列。

当出现下列任何一种条件时，RUNSTATS 将不收集分布统计信息：

- 当将 *num_freqvalues* 配置参数设置为零(0)，以及将 *num_quantiles* 数据库配置参数设置为零(0)或 1 时。
- 当每个数据值是唯一的时候。
- 当该列是 LONG、LOB 或结构化列时。
- 如果列中只有一个非空值。
- 声明的临时表。

下面的例子说明了使用 RUNSTATS 收集目录统计信息以及指定 *num_freqvalues* 和 *num_quantiles* 的不同方法：假设 *num_freqvalues* 数据库配置参数的值是 10，*num_quantiles* 数据库配置参数的值是 20。

收集包含分布统计信息的数据库统计信息：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
```

将 num_freqvalues parameter 设置为 10，将 num_quantiles parameter 设置为 20，因为在命令行上没有指定 num_freqvalues 和 num_quantiles 参数。

仅收集表上的数据库统计信息，其中使用指定的 num_freqvalues 以及从数据库配置设置选择 num_quantiles 收集所有列上的分布统计信息：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION DEFAULT
  NUM_FREQVALUES 40
```

将 num_freqvalues 参数设置为 40，将 num_quantiles 参数设置为 20。

收集表上的数据库统计信息，包含列 deptno 和 deptname 上的分布统计信息。单独为 deptname 列设置分布统计信息的范围，而 deptno 列使用公共的默认值，并且还为两个索引 IDX1 和 IDX2 收集数据库统计信息：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
   (deptno, deptname NUM_FREQVALUES 50 NUM_QUANTILES 100) DEFAULT
   NUM_FREQVALUES 5 NUM_QUANTILES 10 AND INDEXES db2admin.IDX1, db2admin.IDX2
```

对于 deptname 列，*num_freqvalues* 是 50，*num_quantiles* 是 100；对于 deptno 列，*num_freqvalues* 是 5，*num_quantiles* 是 10。

收集所有索引上的数据库统计信息，包含列 deptname 上的分布统计信息。未列出的列

上的分布统计信息将被清除：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
   (deptname NUM FREQVALUES 20 NUM QUANTILES 40)
   AND INDEXES ALL
```

对于 deptname 列，*num_freqvalues* 是 20，*num_quantiles* 是 40。

收集所有索引以及列 deptno 和 deptname 上的数据库统计信息。deptno 列的 num_freqvalues 和 num_quantiles 值将从默认值中获得：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
  (deptno, deptname NUM FREQVALUES 20 NUM QUANTILES 40) DEFAULT
   NUM_FREQVALUES 0 NUM_QUANTILES 0  AND INDEXES ALL
```

对于 deptname 列，*num_freqvalues* 是 20，*num_quantiles* 是 40。对于 deptno 列，*num_freqvalues* 是 0，*num_quantiles* 是 0。其他所有列不包含任何统计信息。

11.2.4　包含列组统计信息的 RUNSTATS

列组(Column Group)统计信息将获得一组列的不同值组合的数目。通常，DB2 优化器可用的基本统计信息不检测数据相关性。列组的使用将给多个谓词的联合选择提供更准确的估计。列组统计信息假设数据是均匀分布的，但还无法获得列组上的分布统计信息。

与列组的基数相比，单个列的基数(cardinality)的乘积将获得更好的相关性估计。

下面的例子说明了如何使用 RUNSTATS 收集捕获了列组信息的数据库统计信息：

```
RUNSTATS ON TABLE db2admin.department ON COLUMNS ((deptno, deptname),
   deptname, mrgno, (admrdept, location))
```

本例中总共有两个列组：(deptno, deptname)和(admrdept, location)。

11.2.5　包含 LIKE STATISTICS 的 RUNSTATS

当在 RUNSTATS 中指定 LIKE STATISTICS 子句时，将收集附加的列统计信息。这些统计信息存储在 SYSIBM.SYSCOLUMNS 表里的 SUB_COUNT 和 SUB_DELIM_LENGTH 列中。它们仅针对字符串列进行收集，查询优化器用它们来提高“column LIKE '%abc'”和“column LIKE '%abc%'”类型谓词的选择性估计。

下面的例子说明了如何使用 RUNSTATS 收集包含 LIKE 统计信息的数据库统计信息：

```
   RUNSTATS ON TABLE db2admin.department ON ALL COLUMNS and COLUMNS (deptname
LIKE STATISTICS)
```

11.2.6 包含统计信息配置文件的 RUNSTATS

从 DB2 V8.2 开始，可以为 RUNSTATS 建立统计信息的配置文件。统计信息配置文件是指一组选项，它们预先定义了特定表上将要收集的统计信息。

当将命令参数“SET PROFILE”添加到 RUNSTATS 命令时，将在表描述符和系统目录中注册或存储统计信息配置文件。如果要更新统计信息配置文件，可以使用命令参数“UPDATE PROFILE”。DB2 中没有删除配置文件的选项。

下面的例子展示了如何使用这项功能。

只注册统计信息配置文件，不收集数据库统计信息：

```
    RUNSTATS ON TABLE db2admin.department AND INDEXES ALL SET PROFILE ONLY
```

RUNSTATS 中的子句“SET PROFILE ONLY”不收集统计信息。

注册统计信息配置文件，并执行所存储统计信息配置文件的 RUNSTATS 命令选项来收集目录统计信息：

```
    RUNSTATS ON TABLE db2admin.department AND INDEXES ALL SET PROFILE
```

仅修改现有的统计信息配置文件，不收集任何数据库统计信息：

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
UPDATE PROFILE ONLY
```

修改现有的统计信息配置文件，并执行已更新的统计信息配置文件的 RUNSTATS 命令选项来收集数据库统计信息：

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
UPDATE PROFILE
```

根据前面已注册的统计信息配置文件来查询 RUNSTATS 选项：

```
    SELECT STATISTICS PROFILE FROM SYSIBM.SYSTABLES WHERE NAME = 'DEPARTMENT'
AND CREATOR = 'DB2ADMIN'
```

使用前面已注册的统计信息配置文件来收集数据库统计信息：

```
    RUNSTATS ON TABLE db2admin.department USE PROFILE
```

11.2.7 带有抽样的 RUNSTATS

随着数据库不断地快速增长，由于时间窗口、内存和 CPU 等约束的限制，通过全表扫描来收集数据库统计信息将会变得越来越困难。这时候可以考虑通过数据抽样，即只扫描

表的数据子集来收集数据库统计信息。

如果查询试图预计总的趋势和模式，并且某一误差域(margin of error)内的近似答案足以监测这些趋势和模式，那么数据抽样或许是比全表扫描更好的选择。

从 DB2 V8.1 开始，引入了 SAMPLED DETAILED 子句，允许通过抽样计算详细的索引统计信息。该子句的使用将减少为获得详细索引统计信息而执行的后台计算量和所需的时间，但在大多数情况下，都会使数据足够精确。

以下是一些关于该子句用法的例子。

收集两个索引上的详细数据库统计信息，但对每个索引条目使用抽样来代替执行详细的计算：

```
    RUNSTATS ON TABLE db2admin.department AND SAMPLED DETAILED INDEXES ALL
```

收集索引上的详细抽样统计信息和表的分布统计信息：

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON KEY
      COLUMNS AND SAMPLED DETAILED INDEXES ALL
```

在 DB2 V8.2 中，提供了对表数据进行抽样的两种新方法：行级的贝努里(Bernoulli)抽样和系统页级的抽样。

11.2.8　带有系统页级抽样的 RUNSTATS

系统页级抽样与行级抽样类似，除了抽样的对象是页面而不是行。以 P/100 的概率选择每一页，而以 1-P/100 的概率拒绝页的选择。在所选中的每一页中，要选择所有的行。系统页级抽样优于全表扫描或贝努里(Bernoulli)抽样的地方在于节省了 I/O。

抽样页也是预取的，所以该方法将比行级贝努里(Bernoulli)抽样更快。与不进行抽样相比，页级抽样极大地提高了性能。

RUNSTATS repeatable 子句允许通过 RUNSTATS 语句生成相同的样本，只要表数据没有发生更改。为了指定该选项，用户还必须提供整数来表示将用于生成样本的种子(seed)。通过使用相同的种子，可以生成相同的样本。

总之，统计信息的准确性取决于抽样率、数据倾斜(data skew)以及用于数据抽样的数据群集。

下面是一些使用贝努里(Bernoulli)行级和系统页级抽样的 RUNSTATS 例子。

收集统计信息，包含 10%行上的分布统计信息：

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION TABLESAMPLE
BERNOULLI (10)
```

为了控制收集统计信息的样本集，以及可以重复使用相同的样本集，需要使用下列语句：

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION TABLESAMPLE
   BERNOULLI (10) REPEATABLE (1024)
```

收集 10%的数据页上的索引统计信息和表统计信息。请注意，只对表数据页进行抽样，而不是索引页。本例中，10%的表数据页用于表统计信息的收集，而对于索引统计信息，将使用所有的索引页：

```
RUNSTATS ON TABLE db2admin.department AND INDEXES ALL TABLESAMPLE SYSTEM
(10)
```

11.2.9 收集统计信息的其他可供选择的方法

对于数据库中的所有表，收集统计信息的一种可供选择的方法就是发出一条 REORGCHK 命令。

使用 REORGCHK 收集所有表的数据库统计信息：

```
REORGCHK UPDATE STATISTICS ON TABLE ALL
```

REORGCHK 命令的 update statistics 选项的作用类似于调用 RUNSTATS 例程来更新数据库中所有表的数据库统计信息，但是所有列上的统计信息只能通过默认的 RUNSTATS 选项来收集。通过使用该命令，还可以收集单个表上或同一模式下一组表的统计信息。

使用 REORGCHK 收集表的数据库统计信息：

```
REORGCHK UPDATE STATISTICS ON TABLE db2admin.department
```

使用 REORGCHK 收集模式的数据库统计信息：

```
REORGCHK UPDATE STATISTICS ON SCHEMA systools
```

虽然该方法看上去是一种在多个表上收集数据库统计信息的快速方法，但是仅仅当 REORGCHK 可以在合理的时间内完成执行时，该方法才是可取的。

在表的 LOAD REPLACE 期间以及索引的创建期间，也可以进行数据库统计信息的收集。

在 UDB V8.2 中，为了在 LOAD 期间收集数据库统计信息，必须将选项“STATISTICS USE PROFILE”添加到 LOAD 控制语句中。选项“STATISTICS YES”仍然有用，但它现在已是过时的语法。如果在表的 LOAD 之后在同一表上执行 RUNSTATS，那么在 LOAD 期间发出“STATISTICS USE PROFILE”的目的就是为了减少正常运行过程中占用的时间。在执行 LOAD 之前，就必须创建统计信息配置文件，现在允许指定与 RUNSTATS 命令中

相同的统计信息选项。

使用 LOAD 命令和 STATISTICS USE PROFILE 子句收集数据库统计信息：

```
LOAD FROM inputfile.del OF DEL REPLACE INTO db2admin.department STATISTICS
USE PROFILE
```

在大多数情况下，RUNSTATS 是在创建索引之后执行的。然而，在执行索引创建操作时，也能收集索引的统计信息。这将避免为了收集统计信息而进行的另一项索引扫描。下面的例子说明了完成这项工作可以使用的一些选项。

收集基本的索引数据库统计信息：

```
CREATE INDEX db2admin.inx1 ON db2admin.department (deptno) COLLECT
STATISTICS
```

收集扩展的索引数据库统计信息：

```
CREATE INDEX db2admin.inx2 ON db2admin.department (deptname) COLLECT
DETAILED STATISTICS
```

收集扩展的索引数据库统计信息，指定使用抽样：

```
CREATE INDEX db2admin.inx3 ON db2admin.department (deptname) COLLECT
SAMPLED DETAILED STATISTICS
```

11.2.10　RUNSTATS 总结

理解并正确使用 RUNSTATS 是维护高性能数据库的关键之一。本节讨论了 RUNSTATS 的重要性，并解释了许多使 RUNSTATS 更有用的可用选项，其中一些新选项通过允许建立指定需要为每个表收集哪些统计信息的配置文件，这些新选项简化了管理。本节展示的示例将使你易于使用 RUNSTATS 获得最佳的数据库性能。

11.3　碎片整理

随着数据被不断删除、插入和更新，数据页和索引页会变得越来越零散，数据页和索引页的物理存储顺序不再匹配其逻辑顺序，数据和索引结构的层次会变得过大，这些都会导致数据页跨越在多个页上和索引页的预读取变得效率低下。因此，根据数据更新的频繁程度需要适当地重新组织表和索引。

11.3.1 表重组(REORG)

对表数据进行大量更改之后，逻辑上连续的数据可能分散存放在很多个不连续的物理数据页中，因此数据库管理器必须执行更多的读操作来访问数据。另外，在某个表删除大量行后，由于表的高水位标记并不会发生变化，因此对该表做全表扫描时就会做很多不必要的 I/O 操作。在这样的情况下，可以考虑重组表以回收浪费的空间和对数据进行重组。此时，既可重组系统目录表，也可以重组数据库表。

注意：

由于重组表的时间通常要比运行统计信息的时间长，因此可以执行 RUNSTATS 以更新当前的数据统计信息并重新绑定应用程序。如果更新的统计信息并没有改善性能，那么重组可能会有所帮助。

1. 重组表的方法

在 DB2 V8 之前要进行重组，只能停止应用，断开连接，离线重组，这对业务的高可用性会造成影响。DB2 V8 后有两种不同的表重组方法：脱机 REORG 以及联机 REORG。

REORG 命令的 INPLACE 选项指定联机重组。如果未指定此选项，那么将运行脱机 REORG。

这两种重组方法各有一些优点和缺点。下面概述了这些优缺点。选择重组方法时，应考虑哪种方法提供的优点是优先要解决的方面以及业务是否允许，例如，如果发生故障时进行恢复比快速完成重组更重要，那么最好使用联机重组方法。

脱机重组的优点：

- 表重组速度最快，尤其在不需要重组 LOB/LONG 数据时。
- 完成时精确地维护了表和索引的集群。
- 重组表之后立即重建索引。不需要单独的步骤来重建索引。
- 可以在临时表空间中构建影子副本，这减少了包含目标表或索引的表空间中需要的空间大小。
- 允许指定并使用集群索引外的索引来重新维护数据的集群，而联机重组在存在集群索引时必须使用现有集群索引。

脱机重组的缺点：

- 表访问受限制。只有在重组的排序和构建阶段才允许应用程序对表进行只读访问。
- 由于使用影子副本方法，因此需要较大空间。

- 与联机 REORG 相比，对 REORG 过程的控制较少：不能在暂停后重新启动脱机重组。

联机重组的优点：

- 允许应用程序在 REORG 期间对表进行完全访问。
- 对 REORG 过程具有更多控制：该过程在后台异步运行，并且可以使其暂停、继续以及停止。例如，如果正在对表进行大量更新或删除操作，那么可以暂停 REORG 过程。
- 在发生故障时可恢复重组过程。
- 由于采用递增方式处理表，因此需要较少空间。

联机重组的缺点：

- 可能产生非最佳数据集群或非最佳索引集群，这取决于 REORG 期间访问表的事务类型。
- 在重组开始时重组的页可能更新次数更多，从而比重组过程中稍后重组的表具有更多碎片。
- 速度比脱机重组要慢。对于正常集群 REORG(不仅仅是空间回收)，执行联机重组的时间可能是脱机重组所用时间的 10~20 倍。对于同时正在对其运行应用程序的表，或者对于定义了大量索引的表，此时间可能还要长很多。
- 联机重组是可恢复的过程，但这会导致日志记录要求提高。可能需要极大量的日志空间，最大可为表大小的若干倍。这取决于重组期间移动的行数、对表定义的索引数以及索引的大小。
- 将维护索引而不是进行重建，因此以后可能需要重组索引。

表 11-1 对联机重组和脱机重组进行了比较。

表 11-1　联机重组与脱机重组的比较

特　　征	脱 机 重 组	联 机 重 组
性能	快	慢
完成时数据的集群因子	良好	非最佳集群
并行性(对表的访问)	从不能访问到只读访问	从只读访问到完全访问
数据存储空间要求	非常大	不是非常大
日志记录存储空间要求	不是非常大	非常大
用户控制(暂停和重新启动重组过程的能力)	较少控制	较多控制
可恢复性	完全可恢复或完全不可恢复：成功或失败	可恢复

(续表)

特　征	脱 机 重 组	联 机 重 组
索引重建	进行	不进行
支持所有类型的表	是	否
指定除集群索引外的索引	是	否
使用临时表空间	是	否

2. 监视表重组的进度

有关表重组当前进度的信息将写入数据库活动的历史记录文件中。历史记录文件中包含每个重组事件的记录。要查看此文件，请执行 db2 list history 命令以打开包含所重组表的数据库。

此外，也可使用表快照来监视表重组的进度。不管如何设置“数据库监视器表”开关，系统均会记录表重组监视数据。下面的示例显示了重组期间调用快照监控的输出。

```
db2 get snapshot for tables on testdb
            表快照
第一个数据库连接时间戳记        = 2012-10-27 00:43:16.630753
快照时间戳记                    = 2012-10-27 00:44:16.475355
数据库名称                              = BANK
数据库路径                          = C:\DB2\NODE0000\SQL00002\
输入数据库别名                      = TESTDB
表列表
 表模式                   = ORACLE
 表名                     = ACCOUNT
 表类型                   = 用户
 数据对象页     = 1820
 读取的行                 = 210000
 写入的行                 = 0
 溢出                     = 0
 重组的页             = 0
 表重组信息:
   重组类型         =
        回收
        表重组
        允许读访问
        仅重组数据
   重组索引         = 0
   重组表空间       = 2
   开始时间                   = 2012-10-27 00:44:06.878339
   重组阶段         = 2 - 替换
```

```
        最大阶段            = 2
        阶段开始时间        = 2012-10-27 00:44:07.576298
        状态                = 已完成
        当前计数器          = 0
        最大计数器          = 0
        完成                = 0
        结束时间            = 2012-10-27 00:44:07.594767
```

3. 以脱机方式重组表

以脱机方式重组表是整理表碎片最快的方法。重组可减少表所需的空间量并提高数据访问和查询性能。

标识需要重组的表之后，可以对这些表以及表上的索引执行 REORG 命令。

要使用 CLP 重组表，请执行 REORG TABLE 命令：

```
db2 reorg table test.employee
```

要使用临时表空间 mytemp 重组表，请输入：

```
db2 reorg table test.employee use mytemp
```

要重组表并根据索引 myindex 对行进行重新排序，请输入：

```
db2 reorg table test.employee index myindex
```

要使用 SQL 调用语句重组表，请使用 ADMIN_CMD 过程发出 REORG TABLE 命令：

```
call sysproc.admin_cmd ('reorg table employee index myindex')
```

在重组表之后，应收集有关表的统计信息，以便优化器具有最准确的数据来评估查询访问方案。

4. 脱机表重组的恢复

在重组替换阶段开始之前，脱机表重组是一个完全成功或完全失败的过程。这表示如果系统在排序或构建阶段崩溃，那么重组表操作将回滚，并且不会在崩溃恢复时重新进行该操作，而是必须在恢复后重新发出 REORG TABLE 命令。

如果系统在进入替换阶段后崩溃，那么重组表操作必须完成。这是因为已完成所有重组表工作，原始表可能不再可用。在崩溃恢复期间，需要已重组对象的临时文件，而不是用于排序的临时表空间。恢复操作将完全重新开始替换阶段，并且需要恢复副本对象中的所有数据。在这种情况下，SMS 表空间与 DMS 表空间之间有如下差别：必须将 SMS 对象从一个对象复制到另一个对象；而在 DMS 表空间中，如果重组在相同表空间中进行，那

么仅指向刚刚重组的对象并废弃原始表。将不重建索引，但在崩溃恢复期间会将索引标记为无效。数据库将根据一般规则确定重建索引的时间：在数据库重新启动时或第一次访问索引时(可以通过设置 DBM 配置参数 INDEXREC 来制定索引重新创建时间和索引重新构建的时间，默认是 RESTART)。

索引重建阶段出现崩溃表示我们已经拥有新对象，因此不重新执行任何操作。如上所述，将不重建索引，但在崩溃恢复期间会将索引标记为无效。数据库将根据一般规则确定重建索引的时间：在数据库重新启动时或第一次访问索引时。

5. 提高脱机表重组的性能

脱机表重组的性能在很大程度上由数据库环境的特征决定。

事实上，以 NO ACCESS 方式运行的重组表操作与以 ALLOW READ ACCESS 方式运行的重组表操作在性能方面没有任何差别。唯一的区别在于，对于 READ ACCESS 方式，DB2 在替换表之前升级对该表的锁定，因此实用程序可能必须等到现有扫描完成并释放其锁定后才能获得相应的锁。在这两种方式下，表在重组表操作的索引重建阶段不可用。

6. 提高重组性能的技巧

如果表空间中有足够的空间，那么对原始表和表的已重组副本使用相同的表空间，而不使用临时表空间。这将节省从临时表空间中复制表所用的时间。

- 考虑在重组表之前删除不必要的索引，以便在重组表操作期间维护较少的索引。
- 确保正确设置了已重组的表所在的表空间的预取大小。
- 启用 INTRA_PARALLEL，以便使用并行处理完成索引重建。
- 调整 sortheap 和 sheapthres 数据库配置参数以控制排序空间。因为每个处理器都将执行私有排序，所以 sheapthres 的值至少应为 sortheap×使用的处理器数。
- 通过调整页清除程序的数目，确保尽快从缓冲池中清除脏索引页。

7. 联机表重组

联机表重组允许用户重组表的同时允许对该表进行完全访问。虽然联机表重组允许用户仍能对数据继续进行访问，但重组速度比脱机表重组要慢。

进行联机表重组时，不是立即重组整个表，而是按顺序重组表的各个部分。不会将数据复制到临时表空间：在现有表对象中移动行以重新建立集群、回收可用空间并消除溢出行。

联机表重组包括 4 个主要阶段：

(1) 选择 N 页(SELECT N pages)。在此阶段中，DB2 将选择 N 页，其中的 N 是包含要进

行重组表处理的扩展数据块(extent)大小(最少 32 个连续页的空间)。

(2) 释放空间(vacate the range)。选定 N 页后，联机表重组将第一阶段选定范围内的所有行移至表中的可用页。每个被移动的行都保留一条 RP(REORG TABLE Pointer)记录，该记录包含行的新位置的 RID。将行作为包含数据的 RO(REORG TABLE Overflow)记录插入到表的可用页中。

重组表完成移动一组行后，将等待表中进行的所有现有数据访问(例如，通过当前执行的应用程序)完成。这些现有访问称为旧扫描程序，它们在访问表数据时使用旧 RID。在此等待过程中启动的任何访问都称为新扫描程序，它们使用新的 RID 来访问数据。在所有旧扫描程序都完成后，重组表操作将清除已移动的行、删除 RP 记录并将 RO 记录转换为正常记录。

(3) 填充空间(fill the range)。所有行的空间都空出后，将采用已重组的格式、根据使用的任何索引进行排序后的顺序并遵循定义的任何预取限制写回这些行。填充范围中的所有页后，将在表中选择下一个 N 有序页，并且循环开始该过程。

(4) 截断表(truncate table)。重组表中的所有页后，默认情况下将移动表的高水位标记以回收空间。如果指定 NOTRUNCATE 选项，那么不会移动表的高水位标记。

8. 联机表重组期间创建的文件

联机表重组期间，将为每个数据库分区创建.OLR 状态文件。此文件是名为 xxxxyyy.OLR 的二进制文件，其中 xxxx 是池标识，而 yyyy 是十六进制格式的对象标识。此文件包含从暂停状态继续联机重组所需的信息。

状态文件包括下列信息：

- 重组类型。
- 所重组的表的生存 LSN。
- 要腾出的下一个范围。
- 重组是为了维护数据的集群还是仅回收空间。
- 用于维护数据集群的索引的标识。

校验和保存在.OLR 文件中。如果该文件已损坏并导致校验和错误，或者如果表 LSN 与生存 LSN 不匹配，那么必须启动新的重组，并且将创建新的状态文件。

如果删除.OLR 状态文件，那么重组表过程就不能继续，并且会返回 SQL2219 错误。必须启动新的重组表过程。

不应从系统中手动删除与重组过程关联的文件。

9. 恢复失败的联机表重组

通常，磁盘已满或日志记录错误之类的处理错误会导致联机表重组失败。如果联机表重组失败，那么 SQLCA 消息将写入历史记录文件中。

如果分区数据库环境中的一个或多个数据库分区遇到错误，那么返回的 SQL 代码是来自报告错误的第一个节点的 SQL 代码。

如果运行时出现故障，那么联机表重组将暂停并进行回滚。如果系统宕机，那么在重新启动时，将开始进行崩溃恢复，并且会暂停并回滚重组。稍后，可以通过使用 REORG TABLE 命令并指定 RESUME 选项来继续重组。由于完整地记录了联机表重组过程，因此可以保证重组过程可恢复。

例如，在某些情况下，联机表重组可能超过 *num_log_span* 限制。在这种情况下，DB2 将强制执行表重组并使其处于暂停状态。在快照输出中，REORG TABLE 命令的状态将显示为“已暂停”。

联机表重组暂停是由输入驱动的，这意味着可以由用户(通过使用 REORG TABLE 命令的暂停(pause)选项或强制执行应用程序命令)或 DB2 在某些情况下(例如在系统崩溃时)暂停。

10. 暂停并重新启动联机表重组

用户可以暂停并重新启动正在进行的联机表重组。

要暂停联机表重组，请发出使用 PAUSE 选项的 REORG TABLE 命令：

```
db2 reorg table homer.employee inplace pause
```

要重新启动已暂停的联机表重组，请发出带有 RESUME 选项的 REORG TABLE 命令：

```
db2 reorg table homer.employee inplace resume
```

注意：

- 暂停联机表重组后，不能开始对该表进行新的重组。必须继续暂停的重组，或者停止暂停的重组，然后再开始新的重组过程。
- 发出 RESUME 请求后，重组过程将沿用在原始 REORG 命令中指定的 TRUNCATE 选项，而无论对后续 START 或任何中间 RESUME 请求指定什么 TRUNCATE 选项。但是，如果重组处于截断阶段并且用户发出指定了 NOTRUNCATE 的 RESUME 请求，那么不会截断表并且重组将完成。
- 在复原和前滚操作后，重组不能继续。

11. 联机表重组的锁定和并行性注意事项

在做联机表重组时，需要考虑的一个重要方面是如何控制锁定，因为这对于应用程序的并行性非常重要。

在联机表重组过程中的某个给定时间点，操作可能拥有下列锁定：

- 为了确保能够对表空间进行写访问，获取受重组表操作影响的表空间的 IX 锁定。
- 在整个重组表操作期间获取并挂起表锁定。锁定级别取决于重组期间允许对该表使用的访问方式：
  - 如果指定了 ALLOW WRITE ACCESS，那么将获取对表的 IS 锁定。
  - 如果指定了 ALLOW READ ACCESS，那么将获取对表的 S 锁定。
- 在截断阶段，当重组操作将行移出截断范围时，将请求对表的 S 锁定，因为旧扫描程序(重组表操作期间存在并访问记录的旧 RID 的数据访问)可能会插入新行。DB2 将等到获取此锁定后再开始截断。就在截断阶段结束时重组表操作对表进行物理截断的那一刻，S 锁定将升级为特殊的 Z 锁定。这表示直到任何现有应用程序都不拥有表锁定(包括来自 UR 扫描程序的 IN 锁定)时，重组表操作才完成。
- 还可以根据表锁定类型获取行锁定：
  - 如果拥有对表的 S 锁定，那么不需要单独的行级别 S 锁定，因此不需要进一步锁定。
  - 如果拥有对表的 IS 锁定，那么在移动行之前将获取行级别 S 锁定，并在移动完成后释放该锁定。
- 可能需要内部锁定来控制对联机表重组对象和其他联机 DB2 实用程序(如联机备份)的访问。

锁定会影响重组表和并发用户应用程序的性能。强烈建议在执行联机表重组时检查锁定快照数据以了解锁定活动。

12. 监视表重组

可以使用 get snapshot 命令、SNAPTAB_REORG 管理视图或 SNAP_GET_TAB_REORG 表函数获取有关表重组操作的状态信息。

要使用 SQL 访问有关重组操作的信息，请使用 SNAPTAB_REORG 管理视图。 例如，以下 SELECT 语句返回有关在当前连接的数据库上对所有数据库分区执行的表重组操作的详细信息：

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
  AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
```

```
    REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
    FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM
TAB_NAME  TAB_SCHEMA   REORG_PHASE   REORG_TYPE    REORG_STATUS REORG_COMPLETION
--------------- --------------- ---------------- -------------------- ----
ACCOUNT      ORACLE      REPLACE    RECLAIM+OFFLINE+ALLO COMPLETED     SUCCESS
  1 条记录已选择
```

如果未重组任何表，那么返回 0 行。

要使用快照监视器访问有关重组操作的信息，请发出 get snapshot for tables on database 命令并检查表重组监视元素(具有前缀“reorg_”)的值。

由于脱机表重组是同步的，因此脱机表重组中出现的任何错误都会返回给实用程序的调用者(应用程序或命令行)。

联机表重组中的错误处理与脱机表重组中的错误处理略有不同。由于联机表重组是异步的，因此没有 SQL 消息写入 CLP。要查看在联机表重组期间返回 SQL 错误，请发出“list history reorg”命令。

联机表重组作为 db2Reorg 进程在后台运行，这意味着从重组表进程成功返回到调用应用程序并不表示重组已成功完成。即使调用应用程序终止数据库连接，db2Reorg 进程也会继续。

11.3.2 索引重组

通过删除和插入操作，在对表进行更新后，索引的性能会降低，表现方式如下：

- 索引叶子页分裂：叶子页被分裂之后，由于必须读取更多的叶子页才能访存表页，因此 I/O 操作成本会增加。
- 物理索引页的顺序不再与这些页上的键顺序相匹配(此称为不良集群索引)。叶子页出现不良集群情况后，顺序预取操作的效率将降低，因此会导致更多的 I/O 等待。
- 形成的索引大于最有效的级别(level)数，在此情况下应重组索引。

如果在创建索引时设置了 MINPCTUSED 参数，那么在删除某个键且可用空间小于指定的百分比时，数据库服务器会自动合并索引叶子页；此过程称为联机索引整理碎片。但是，要复原索引集群和可用空间以及降低索引叶级别，请使用下列其中一种方法：

- 删除并重新创建索引。
- 使用 REORG INDEXES 命令联机重组索引。因为此方法允许用户在重建表索引期间对表进行读写操作，所以在生产环境中可能需要选择此方法。
- 使用允许脱机重组表及其索引的选项运行 REORG TABLE 命令。

1. 联机索引重组

在使用 ALLOW WRITE ACCESS 选项运行 REORG INDEXES 命令时，如果同时允许对指定的表进行读写访问，那么会重建该表的所有索引。进行重组时，对基表所做的任何将会影响到索引的更改都将记录在 DB2 日志中。另外，如果有任何内部内存缓冲区空间可供使用，那么还将这些更改存放在这样的内存空间中。重组将处理所记录的更改以便在重建索引时与当前写活动保持同步更新。内部内存缓冲区空间是根据需要从实用程序堆中分配的指定内存区域，用来存储对正在创建或重组的索引所做的更改。使用内存缓冲区空间使索引重组操作能够通过这样的方式来处理更改，即先直接从内存读取，然后读取日志(如有必要)，但读取日志的时间要晚得多。在重组操作完成后，将释放所分配的内存。重组完成后，重建的索引可能不是最佳集群的索引。如果为索引指定 PCTFREE，那么在重组期间，每页上均会保留相应百分比的空间。

对于分区表，支持对各个索引进行联机索引重组和清除。要对各个索引进行重组，指定索引名：REORG INDEX *index_name* for TABLE *table_name*。

注意：

REORG INDEXES/INDEX 命令的 CLEANUP ONLY 选项不能完全重组索引。CLEANUP ONLY ALL 选项将去除那些标记为“删除”且被认为要落实的键。此外，还将释放所有标记为“删除”且被认为要落实的键所在的页。在释放页后，相邻的叶子页将会合并，前提是这样做可以在合并页上至少留出 PCTFREE 可用空间。PCTFREE 是指在创建索引时为其定义的可用空间百分比。CLEANUP ONLY PAGES 选项仅删除那些标记为“删除”且被认为要落实的所有键所在的页。

使用 CLEANUP ONLY 选项对分区表的索引进行重组时，支持任何访问级别。如果未指定 CLEANUP ONLY 选项，那么默认访问级别 ALLOW NO ACCESS 是唯一支持的访问级别。

索引重组具有下列要求：

- 对索引和表具有 SYSADM、SYSMAINT、SYSCTRL 或 DBADM 权限，或者具有 CONTROL 特权。
- 用于存储索引的表空间的可用空间数量大于或等于索引的当前大小。在发出 CREATE TABLE 语句时，考虑在大型表空间中重组索引。
- 其他日志空间。索引重组需要记录其活动。因此，重组可能会失败，尤其是在系统繁忙和记录其他并发活动时。

注意：

如果具有 ALLOW NO ACCESS 选项的 REORG INDEXES ALL 命令运行失败，那么会标记索引无效并且此项操作不可撤销。但是，如果具有 ALLOW READ ACCESS 选项的 REORG 命令或具有 ALLOW WRITE ACCESS 选项的 REORG 命令运行失败，那么可以复原原来的索引对象。

2. 确定何时重组表和索引

在对表数据进行许多更改之后，逻辑上连续的数据可能会位于不连续的物理数据页上，在许多更新操作创建了溢出(overflow)记录时尤其如此。按这种方式组织数据时，数据库管理器必须执行额外的读操作才能访问顺序数据。另外，在删除大量数据后，空间并没有释放，也需要执行额外的读操作。

表重组操作会通过整理数据碎片来减少浪费的空间，并对行进行重新排序以合并溢出记录，从而加快数据访问速度并最终提高查询性能。还可以指定根据特定索引来重新排序数据，以便查询通过最少的 I/O 读取操作就可以访问数据。

对表数据进行大量更改将导致更新索引并使索引性能下降。索引叶子页可能变成碎片或出现不良集群情况，并且索引有可能形成比所需层次(level)要多的层次以获得最佳性能(通常，几百万条记录的索引层次一般为 3，正常生产环境中索引的层次很少超过 4)。所有这些问题都会产生更多 I/O 并导致性能下降。

下列任何情况都需要我们重组表或索引：

- 自上次重组表之后，对查询访问的表进行了大量的插入、更新和删除活动。
- 对于使用具有高聚合度的索引的查询，其性能发生了明显变化。
- 在执行 RUNSTATS 以刷新统计信息后，性能没有得到改善。
- REORGCHK 命令指示需要重组表或索引(在某些情况下，REORGCHK 始终建议重组表，即使在执行了重组后也是如此)。例如，如果使用 32KB 页大小，并且平均记录长度为 15 字节且每页的最多包含 253 条记录，那么每页具有 32700−(15×253)= 28905 个不可用字节。这意味着大约 88%的页面是可用空间。用户应分析 REORGCHK 的建议并针对执行重组所需的成本平衡利益。
- 综合考虑查询性能不断降低所浪费的成本和重组表所需的成本(包括 CPU 时间、重组的时间和 REORG 实用程序在完成重组操作之前锁定表造成的并行性降低)，以确定是否进行表重组。

要确定是否需要重组表或索引，查询系统目录表中的统计信息并监视下列统计信息：

1)　行的溢出(行链接)

查询 SYSSTAT.TABLES 视图中的 OVERFLOW 列以监视溢出值。当表中的可变长度列导致记录长度变长，以致它们不能放入数据页上的指定位置时，这时行数据就会溢出。在列添加至表定义并稍后通过更新行来更新该列时，长度也可能会更改。在这些情况下，在行中的原始位置将保留一个指针，而实际值存储在由指针指示的另一个位置。这可能会影响性能，因为数据库管理器必须根据指针来查找行的内容。这个包括两个步骤的过程增加了处理时间，并且还会增加需要的 I/O 数目。

重组表数据将消除行溢出；如果行链接数量较多，则重组表数据的效果就会比较明显。

2) 访存统计信息

当以索引顺序访问表时，查询 SYSCAT.INDEXES 和 SYSSTAT.INDEXES 系统目录表中的以下三个列来确定预取程序的效率。这些统计信息对照基表来体现预取程序的平均性能的特征。

- AVERAGE_SEQUENCE_FETCH_PAGES 列存储可以按表中顺序访问的平均页数。可以按顺序访问的页适合于预取。较小的数目指示预取程序没有充分发挥作用，原因是它们不能读入由表空间的 PREFETCHSIZE 设置指定的所有页数。较大的数目指示预取程序将有效地执行。对于集群索引和表，此数目应该接近 NPAGES(包含一些行的页数)的值。
- AVERAGE_RANDOM_FETCH_PAGES 列存储当使用索引来访存表行时在顺序页访问之间的平均随机表页数。当大多数页按顺序排列时，预取程序忽略少量的随机页，并按已配置的预取大小继续预取。当表变得更加混乱时，随机访存页数就会增加。通常是由于在表的末尾或在溢出页中发生了无序的插入，才导致这样的无组织。当使用索引来访问某一范围内的值时，这会导致降低查询性能的访存。
- AVERAGE_SEQUENCE_FETCH_GAP 列存储当使用索引进行访存时表页序列之间的平均间隔。通过扫描索引叶子页来进行检测，每个间隔表示在表页的各个序列之间必须随机访存的平均表页数。当随机访问许多页时会发生这些情况，这会中断预取程序。较大的数目指示无组织的表或较差集群的索引。

3) 包含标记为已删除但未除去的 RID 的索引叶子页数

在 type-2 类型索引中，当 RID 标记为删除时，通常未在物理上删除 RID，这意味着可用空间可能会被这些逻辑上已删除的 RID 占用。要检索每个 RID 都被标记为已删除的叶子页的数目，查询 SYSCAT.INDEXES 和 SYSSTAT.INDEXES 统计信息表的 NUM_EMPTY_LEAFS 列。对于并非所有 RID 都标记为删除的叶子页，逻辑上已删除的 RID 的总数存储

在 NUMRIDS_DELETED 列中。

使用此信息来通过执行带 CLEANUP ALL 选项的 REORG INDEXES 估计可以回收多少空间。要想只回收所有 RID 都被标记为已删除的页中的空间，执行带有 CLEANUP ONLY PAGES 选项的 REORG INDEXES。

4) 索引的聚合比率和聚合因子统计信息

聚合比率统计信息存储在 SYSCAT.INDEXES 系统目录表的 CLUSTERRATIO 列中。此值(在 0~100 之间)表示使用索引的数据聚合的程度。如果收集 DETAILED 索引统计信息，0 和 1 之间好的聚合因子统计信息存储在 CLUSTERFACTOR 列中，并且 CLUSTERRATIO 的值为－1。在这两个集群统计信息中，只有一个可以记录在 SYSCAT.INDEXES 目录表中。要将 CLUSTERFACTOR 值与 CLUSTERRATIO 值进行比较，可以将 CLUSTER- FACTOR 乘以 100 以获得百分比。

注意：

通常，表中只有一个索引可以具有较高的聚合度。

如果查询用到的索引需要访问表中比较多的数据，而不是只访问索引或者只通过唯一索引访问表的一条记录，那么在具有较高聚合比率的情况下可能执行得更好。低的聚合度导致此类扫描要执行更多的 I/O，因为在每个数据页经过第一次访问后，下次访问该页时，该页仍在缓冲池中的可能性减小。增大缓冲区大小也可以提高非聚合索引的性能。

如果表数据最初是对某个索引聚合的，而集群统计信息指示现在很少为同一索引聚合数据，那么你可能想重组该表以再次聚合数据。

5) 索引叶子页数

查询 SYSCAT.INDEXES 表中的 NLEAF 列以便了解索引占用的叶子页数目，该数目告诉你对索引进行完整的扫描需要多少索引页 I/O。

理想情况下，索引应该尽可能占用最小的空间量，以便减少进行索引扫描所需要的 I/O 次数。随机的更新活动可导致页分割，这就增大了索引的大小。当在重组表期间重建索引时，可以使用最小空间量来构建每个索引。

注意：

在默认情况下，当构建索引时，会在每个索引页上保留 10%的可用空间。要增加可用空间量，当创建索引时指定 PCTFREE 参数。无论何时重组索引，都使用 PCTFREE 值。

6) 空数据页的数目

要计算表中的碎片数，查询 SYSCAT.TABLES 中的 FPAGES 和 NPAGES 列并从 FPAGES

数减去 NPAGES 数。FPAGES 列存储正在使用的总页数；NPAGES 列存储包含一些行的页数。当删除整个范围内的行时，可能出现空页。

随着碎片的增多，就更需要进行表重组。重组表时将回收碎片并减少表使用的空间量。另外，因为会将碎片的数据页读入缓冲池以进行表扫描，所以回收未使用的页可以提高表扫描的性能。

REORGCHK 命令返回有关数据组织的统计信息，并且可以在是否需要重组特定表或索引这一问题上提供建议。以下为 REORGCHK 命令的输出：

```
db2 reorgchk update statistics
正在执行 RUNSTATS ....
表统计信息:
F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (数据页的有效空间使用率) > 70
F3: 100 * (需要页数 / 总页数) > 80
SCHEMA.NAME          CARD      OV     NP     FP ACTBLK    TSIZE  F1  F2  F3 REORG
表: ORACLE.ACCOUNT
                     100000      0   1820   1820      -  7200000   0  98 100 ---
表: ORACLE.AUDITLOG
                     3382      0     29     29      -   114988   0 100 100 ---
表: ORACLE.CONNHEADER T1
                     21      0      2      2      -     4158   0 100 100 ---
表: ORACLE.CONTROL T1
                     2      0      1      1      -       84   0   - 100 ---
表: ORACLE.STMT T1
                     403      0     58     58      -   205127   0 100 100 ---
表: SYSTOOLS.HMON ATM INFO
                     136     86      6     13      -    47872  63 100 100 *--
表: SYSTOOLS.HMON COLLECTION
                                  -      -      -      -      -        -   -   -
表: SYSTOOLS.POLICY
                     5      0      1      1      -      740   0   - 100 ---
```

在上面的输出中，我们重点关注“REORG”字段。如果 REORG 字段有一个或多个“*”，那么可以考虑对该表或索引重组。

在重组之前，请综合考虑查询性能不断降低浪费的成本和重组表或索引所需的成本(包括 CPU 时间、耗用时间和 REORG 命令在完成重组操作之前锁定表造成的并行性降低)，以确定是否进行表重组。

11.3.3 重组表和索引的成本

决定是否要重组表或索引时，必须考虑对这些对象执行重组时产生的一些开销。

重组表和索引的成本包括：

- 执行 REORG 命令时消耗的 CPU。
- 运行 REORG 命令时并行性降低。由于重组要求进行锁定而导致并行性降低。
- 需要额外的存储空间(脱机表重组需要额外的存储空间来保存表的影子副本。联机表重组需要更多的日志记录空间。联机索引重组需要额外的存储空间来保存索引的影子副本和更多日志空间。脱机索引重组将使用较少的日志空间并且不涉及影子副本)。

在某些情况下，已重组的表可能比原始表要大，从而相应地增加了空间要求。在下列情况下，重组后表可能会增大：

- 在使用索引来确定行顺序的集群重组表中，如果表记录的长度可变(例如使用 varchar)，那么重组结束时可能会使用更多空间，因为某些页中包含的行数可能比原始表中的行数要少。
- 在重组之前、但在创建之后，在表中添加了列，并且重组之后在某些行中可能是第一次实现添加的列。
- 自上次重组后，每页上剩余的可用空间大小(由 PCTFREE 属性的值表示)已增加。
- 表包含 LOB，并且该 LOB 可能比以前使用更多的空间。

1. 脱机表重组的空间要求

由于脱机重组使用影子副本方法，因此需要足够的额外存储空间来容纳表的另一个副本。在原始表所在的表空间中或用户指定的临时表空间中构建影子副本。

如果使用表扫描排序，那么可能需要其他临时表空间来进行排序处理。需要的其他空间可能与要重组的表一样大。如果集群索引是 SMS 类型或唯一的 DMS 类型，那么重新创建此索引不需要排序。但是，将通过扫描刚刚重组的数据来重建此索引。任何其他需要重新创建的索引都将涉及排序，并可能需要多达要重组的表大小的临时表空间。

脱机表重组生成少量控制日志记录，因此消耗相对较少的日志空间。如果重组未使用索引，那么唯一的日志记录是表数据日志记录。如果指定了索引，或者如果表上存在集群索引，那么按照将行放入新版本的表中的顺序记录这些行的 RID。每条 RID 日志记录最多包含 8000 个 RID，每个 RID 消耗 4 字节。这可能是导致在脱机表重组期间用完日志空间的因素之一。请注意，仅当数据库可恢复(LOGRETAIN=ON)时才记录 RID。

2. 联机表重组的日志空间要求

联机表重组所需的日志空间通常比脱机表重组所需的日志空间要大。需要的空间大小由要重组的行数、索引数、索引键的大小以及最开始表的组织情况决定。为用于表的日志空间确定典型的基准是一种不错的做法。

表中的每行都可能在联机表重组期间移动两次。如果提供索引，那么每行必须更新索引键以反映新位置，并且在完成对旧位置的所有访问后，再次更新索引键以除去对旧 RID 的引用。移回行时，将再次执行对索引键的这些更新。将记录所有这些活动以便联机表重组完全可恢复，因此最少应有两条数据日志记录(包括行数据的每个实例)和 4 条索引日志记录(包括键数据的每个实例)。集群索引尤其容易填满索引页，从而导致索引分裂和合并，这些分裂和合并活动也必须进行记录。

由于联机表重组经常发出内部落实，因此通常不会拥有重要的活动日志。如果在某个时间联机重组拥有大量活动日志，那么必定是在截断阶段，因为截断阶段需要 S 表锁定。如果表重组无法获取该锁定，那么将等待并保留日志，而其他事务可能会快速填满日志。

11.3.4　合理设计以减少碎片生成

可以使用不同的策略来降低需要重组表和索引的频率，从而避免增加不必要的重组操作的成本。

1. 减少重组表的需要

- 使用多分区表。表越小，需要重组的可能性就越小。
- 创建多维集群(MDC)表，将在 CREATE TABLE 语句的 ORGANIZE BY DIMENSION 子句中指定的列中自动维护表的集群。
- 对表打开 APPEND 方式。例如，如果这些新行的索引键值总是新的更大的键值，那么表的集群属性将尝试将其放置在表的末尾。在这种情况下，将表置于追加方式(append on)可能优于集群索引。
- 在创建表之后：
    - ◊ 改变表以添加 PCTFREE。
    - ◊ 在索引上使用指定的 PCTFREE 创建集群索引。
    - ◊ 在将数据装入表中之前对数据进行排序。
    - ◊ 装入数据。

表上正确设置了 PCTFREE 的集群索引有助于保持原始排序顺序。当表页上有足够的空间时，可以将新数据插入正确的页以维护索引的集群特征。随着更多的数据被插入，表

页会因此而变满，记录会被追加至表的末尾，因而表将逐渐失去集群特性。

如果在创建集群索引后执行 REORG TABLE 命令，或者执行排序和 LOAD 命令，那么索引会尝试维护数据的特定顺序，这将改善 RUNSTATS 实用程序收集的 CLUSTERRATIO 或 CLUSTERFACTOR 统计信息。

注意：

创建多维集群(MDC)表可以减少重组表的必要性。对于 MDC 表，将在 CREATE TABLE 语句的 ORGANIZE BY DIMENSIONS 子句中指定为参数的列上维护集群。

2. 减少重组索引的需要

- 在索引页上使用 PCTFREE 或 LEVEL2 PCTFREE 创建集群索引。范围在 0~99%之间，默认值为 10%。
- 使用 MINPCTUSED 创建索引。可能的范围在 0~99%之间，建议的值为 50%。
- 考虑使用 REORG INDEXES 命令的 CLEANUP ONLY ALL 选项来合并叶子页。

11.3.5 启用表和索引的自动重组

在 DB2 V8 之前，我们对表和索引做碎片整理时，只能通过写脚本方式定期在业务逻辑许可的情况下对表和索引做碎片整理。在 DB2 V8 之后，可以使用自动表重组来启用 DB2 管理脱机和索引重组，这使得不用再担忧何时使用何种方法来重组数据。可启用 DB2 来重组系统目录表以及数据库表。

可以使用图形用户界面工具或命令行界面来打开此功能。

- 要使用图形用户界面工具来将数据库设置为自动重组：

(1) 通过以下两种方法之一来打开“配置自动维护”向导：一种方法是在“控制中心”中右键单击数据库对象；另一种方法是在“运行状况中心”中右键单击要配置自动重组的数据库实例。从弹出窗口中选择**配置自动维护**。

(2) 在此向导中，可以启用自动重组来整理数据碎片、指定想要自动重组的表以及指定用于执行 REORG 实用程序的维护窗口。

- 要使用命令行界面来将数据库设置为自动重组，请将下列配置参数设置为“ON”：
  - ◊ AUTO_MAINT
  - ◊ AUTO_TBL_MAINT
  - ◊ AUTO_REORG

不过不建议使用自动表重组，因为自动表重组是数据库控制的，我们不能干预，很可能在业务高峰期间开始了自动表重组，这会影响性能。所以，建议在充分了解业务逻辑的

情况，通过编写 crontab 脚本，在合适的时间调度执行碎片整理。

11.4　碎片整理案例分析

11.4.1　执行表、索引检查是否需要做 REORG

首先，执行表、索引检查是否需要做碎片整理：

```
    ---------------------------------------------DB2
CLP-----------------------------------------
db2 reorgchk update statistics on table db2admin.employee
    执行 RUNSTATS ....
    表统计信息：
    F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (Effective Space Utilization of Data Pages) > 70
F3: 100 * (Required Pages / Total Pages) > 80
    SCHEMA     NAME                   CARD     OV     NP     FP
ACTBLK     TSIZE  F1   F2   F3 REORG
---------------------------------------------------------------------------
DB2ADMIN  EMPLOYEE          258500 51699 12932 16165      -
61781500  19  93  80 *-*
---------------------------------------------------------------------------
    索引统计信息：
    F4: CLUSTERRATIO 或正常化的 CLUSTERFACTOR > 80
F5: 100 * (KEYS * (ISIZE + 9) + (CARD - KEYS) * 5) / ((NLEAF - NUM EMPTY LEAFS)
* INDEXPAGESIZE) > 50
F6: (100 - PCTFREE) * ((INDEXPAGESIZE - 96) / (ISIZE + 12)) ** (NLEVELS - 2)
* (INDEXPAGESIZE - 96) / (KEYS * (ISIZE + 9) + (CARD - KEYS) * 5) < 100
F7: 100 * (NUMRIDS DELETED / (NUMRIDS DELETED + CARD)) < 20
F8: 100 * (NUM EMPTY LEAFS / NLEAF) < 20
    SCHEMA    NAME                    CARD  LEAF ELEAF  LVLS
ISIZE  NDEL   KEYS  F4  F5  F6  F7  F8 REORG
---------------------------------------------------------------------------
表：DB2ADMIN.EMPLOYEE
DB2ADMIN IDX_EMP_C            258500 14894     0     4   106 21040
258500  72  48  13   7   0 **---
SYSIBM  SQL060417152213950 258500  7122     0     4    60     0
258500  72  61  62   0   0 *----
```

CLUSTERRATIO 或正常化的 CLUSTERFACTOR(F4)将指示索引需要 REORG，该索

引与基本表不在相同的序列中。当在表中定义了多个索引时，一个或多个索引可能被标记为需要 REORG。指定 REORG 顺序的最重要索引。

使用 ORGANIZE BY 子句和相应的维索引定义的表的名称有'*'后缀。维索引的基数等价于表的“活动的块数”统计信息。

DB2 提示信息说明

对 REORGCHK 所使用的度量的考虑因素包括(当查看 REORGCHK 工具的输出时，找到用于表的 F1、F2 和 F3 这几列，以及用于索引的 F4、F5、F6、F7 和 F8 这几列。如果这些列中的任何一列有星号(*)，就说明当前的表和/或索引应该重组)：

F1：属于溢出记录的行所占的百分比。当这个百分比大于 5%时，在输出的 F1 列中将有一个星号(*)。

F2：数据页中使用了的空间所占的百分比。当这个百分比小于 70%(也就是说有 30%左右的碎片)时，在输出的 F2 列上将有一个星号(*)。

F3：其中含有包含某些记录的数据的页所占的百分比。当这个百分比小于 80%时，在输出的 F3 列上将有一个星号(*)。

F4：群集率，即表中与索引具有相同顺序的行所占的百分比。当这个百分比小于 80%时，在输出的 F4 列上将有一个星号(*)。

F5：在每个索引页上用于索引键的空间所占的百分比。当这个百分比小于 50%时，在输出的 F5 列上将有一个星号(*)。

F6：可以存储在每个索引级的键的数目。当这个数字小于 100 时，在输出的 F6 列上将有一个星号(*)。

F7：在一个页中被标记为 deleted 的记录 ID(键)所占的百分比。当这个百分比大于 20%时，在输出的 F7 列上将有一个星号(*)。

F8：索引中空叶子页所占的百分比。当这个百分比大于 20%时，在输出的 F8 列上将有一个星号(*)。

11.4.2 表和索引碎片整理

(1) 针对 REORGCHK 给出的提示信息(特别是打*号的 REORG 列)，结合 SQL 语句本身的构成，建立适当的索引。

(2) 根据实际情况，重组表、重组索引。

(3) 更新表、索引统计信息。

(4) 如果应用程序是静态 SQL，重新绑定(rebind)应用程序包。

例如：

```
-------------------------------DB2 CLP---------------------------------
db2 reorg table db2admin.employee;
db2 reorgchk update statistics on table db2admin.employee;
db2 reorg indexes all for table db2admin.employee;
db2 runstats on table db2admin.employee and indexes all;
```

11.5 案例：生成碎片检查、统计信息更新、碎片整理和 REBIND 脚本

在实际生产中，可以编写统计信息更新、碎片整理和重新绑定脚本来调度统计信息更新，脚本如下：

```
    db2 connect to sample
    db2 “select 'reorg  table '||rtrim(tabschema||'.'||tabname||';' from
syscat.tables where type='T' > reorg.sql
    db2 “select 'reorg indexes all for table
'||rtrim(tabschema||'.'||tabname||';' from syscat.tables where type='T' >
reorg index.sql
    db2 “select 'runstats on table '||rtrim(tabschema||'.'||tabname||' and
indexes all;' from syscat.tables where type='T' > runstats.sql
    db2 “select 'rebind package '||rtrim(pkgschema||'.'||pkgname||';' from
syscat.packages where pkgschema not in ('NULLID' ) > rebind.sql
```

把生成的结果分别编辑，存放到特定目录下。

在 UNIX 上编写 shell 脚本 maint.sh：

```
While 1>0 do
db2 -tvf reorg.sql -z reorg error.log
db2 -tvf reorg index.sql -z reorg index.log
db2 -tvf runstats.sql -z runstats error.log
db2 -tvf rebind.sql -z reinbd_error.log
```

然后用 crontab 调度在合适的时间(业务最闲的时候)执行，例如：

```
#每天早上 3 点执行一次 /bin/ls :  0 3 * * * /home/db2inst1/sqllib/mon/maint.sh
```

11.6 重新绑定程序包

重新绑定(REBIND)是为先前绑定的应用程序重新创建程序包的过程。每个在数据库中

执行的应用程序在执行之前都需要有一个绑定过程，这个绑定过程会根据数据库中各种统计信息和相关数据库对象的情况创建一个程序包，这个程序包中通常就是执行计划。所以在统计信息或相关数据库对象被修改之后，就需要对数据库中受影响的应用程序执行重新绑定，这样才能允许应用程序只用最新的更新。

那么具体哪些程序包必须执行重新绑定呢？DB2 的系统表 SYSCAT.PACKAGES 的 VALID 列标识当前的程序包是否可用，如果此列的值为 X，就表示当前的程序包是不可用的，此程序包就需要被重新绑定。这些程序包可以使用下面的方法进行绑定，也可以让数据库管理器在执行这样的程序包时自动完成重新绑定(数据库管理器默认会在隐式执行这些不可用的程序包时重新将它们绑定)。

建议在对统计信息更新之后，对数据库中的应用程序执行重新绑定以更新执行计划。还有一些其他情况需要及时重新绑定程序包，比如创建了新的索引等。统计信息更新、碎片整理和重新绑定的顺序如图 11-3 所示。

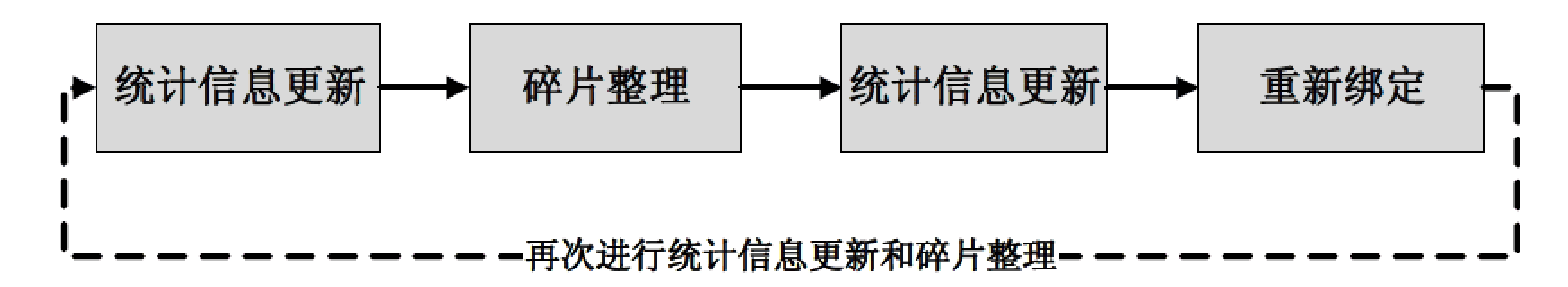


图 11-3　统计信息更新和碎片整理的处理流程

有两个命令可以执行重新绑定：

- REBIND
- db2rbind

1. REBIND 命令

REBIND 命令可以实现重新创建数据库中已经存在的程序包，具体语法如下：

```
>>-REBIND--+---------+--package-name--------------------------->
           '-PACKAGE-'
>--+-----------------------+--RESOLVE--+-ANY----------+--------->
   '-VERSION--version-name-'           '-CONSERVATIVE-'
>--+--------------+--------------------------------------------><
   +-REOPT NONE---+
   +-REOPT ONCE---+
   '-REOPT ALWAYS-'
```

2. db2rbind 命令

db2rbind 命令可以重新绑定数据库中存在的所有程序包，具体语法如下：

```
>>-db2rbind--database-- -l--logfile--+-----+-------------------->
                                      '-all-'
>--+---------------------------+--+----------------------+--->< 
   '- -u--userid-- -p--password-'  |     .-conservative-. |
                                   '- -r--+-any----------+-'
```

此命令的功能类似于 REBIND 命令，只是更简单一些，而且可以批量处理数据库中所有的程序包。

11.7　DB2 健康检查

在笔者管理数据库的经历中，很多时候是业务部门的人反馈说系统反应慢，或者登录不上，这时候去检查数据库，经常是发现数据库已经对任何操作都没有响应、或是数据库 hang 住了、甚至数据库已经宕机了，这个时候 DBA 就变得很被动，影响也不好。DBA 经常很困惑，什么样的数据库才算是性能很棒的数据库呢？有没有维护指标能反映出数据库可能有潜在的问题？性能监控的定位不应该是出现问题的时候去找问题原因，而是对数据库的日常监控、监控检查和日常维护，目标是及时发现数据库潜在问题。这和问题诊断是不一样的，就像一个健康的人，也要定期体检，而不只是等到生病后再治病。

11.7.1　查看是否有僵尸实例进程

在 Linux/UNIX 上可以通过 db2_ps 或 ps –ef | grep –i instname 来查看实例的所有进程，应该检查是否存在僵尸(zombie)的 defunct 实例进程。

在 UNIX 的进程结构中，每一个进程都有父进程。当一个进程结束时会通知它的父进程，从而该进程的父进程会收集该进程的状态信息。如果父进程在一定的时间内无法收集到状态信息，那么系统中就会残留 defunct 进程。因为 defunct 进程是已经停止的，所以使用杀死进程的方法来杀 defunct 进程是无效的。defunct 进程不使用 CPU 或硬盘等系统资源，而只使用极少量的内存用于存储退出状态和资源使用信息。下面是检查 DB2 进程中是否有僵尸进程的例子：

```
# ps -emo THREAD | grep -i Z | grep -i db2inst1
db2inst1 1716 19292 - Z 10 60 1 * 260801  - db2agent  -----注：状态 Z 表示
                                                            僵尸(zombie)进程
- - - 4863 S 0 60 0 599e9d8 8400 - - -
```

```
- - - 5537 R 10 60 1 5999e18 2420 - 3 -
db2inst1 19292 18524 - Z 0 60 0 586ad84 200001 - -db2agent
- - - 7617 S 0 60 0 586ad84 400 - - -
db2inst1 25864 31168 - Z 11 65 0 - 200001  - -db2agent
- - - 8993 R 11 65 0 - 0 - - -
```

11.7.2 检查数据库是否一致

关于一致性(consistent)的定义容易混淆，而且 get db cfg 命令的报告方式也常常会产生歧义。

按照定义，对于数据库来说，如果提交的所有事务都已经写到了磁盘上，并且任何未提交的事务都不在磁盘上，那么数据库就是一致的。当数据库正在运行的时候，如果有应用程序连接到数据库，那么就会有一些对页做了更改的事务，也许这些事务已经被提交，但是被更改的页可能还没有更新到磁盘上。在这种情况下，get db cfg 将报告数据库是不一致的，但实际上该数据库完全没问题。因此，仅仅获得关于数据库的所有数据库配置信息是不够的。

一个好方法是可以通过 inspect 命令检查数据库是否一致，如下所示：

```
db2 inspect check database results keep db_check.out
DB20000I  INSPECT 命令成功完成
db2inspf db_check.out db_check.txt
最后检查 db_check.txt 文件，查看数据库是否一致
```

11.7.3 查找诊断日志以判断是否有异常

错误和消息日志会被分散记录到两个日志文件——db2diag.log 和 instance_ID.nfy 中。通知日志(instance_ID.nfy)包含用于 DBA 的消息；而 db2diag.log 文件则在需要报告关于 DB2 的问题时，由 DBA 用来定位、诊断和分析问题。DBA 应该经常检查 db2diag.log 和通知日志，查看是否有新的错误、警告或严重的事件。

DB2 提供了 db2diag 工具，用于对 db2diag.log 中提供的大量信息进行过滤和格式化，详细的命令用法请参考 DB2 信息中心。

11.7.4 检查数据库备份完整性、日志归档是否正常

我们一般会使用第三方存储管理软件，比如 TSM ，把归档日志和备份文件映像从磁盘转移到磁带上。设置的方法有很多种，比较常见的有：写一个 shell 脚本，放到操作系统 crontab 中定期执行。shell 脚本中写下了多长时间使用 TSM 进行备份等操作。作为 DBA，我们需要经常检查(或由其他相关管理人员告知)磁带机是否运行正常，第三方管理软件是否运行正常，备份映像文件是否正常放到了磁带上，归档日志是否正常放到了磁带上等等。

只有在保证这些正常的情况下，我们才能保证备份映像文件和归档日志都被正确处理了；当数据库发生故障的时候，我们才能利用备份映像文件和归档日志进行数据库的还原和恢复。如果数据库是只读的，或者很容易从头开始重建，那么很可能不会启用主日志归档方法(logarchmeth1)，所以可以略过这一步骤。然而，对于那些 OLTP 事务处理数据库来说，由于丢失任何提交的事务都是承受不起的，因此确保主日志归档方(logarchmeth1)功能处于启用状态，并且日志可以成功地归档。因为这样可以保证在出现灾难的时候重建数据库，并让事务重演。

虽然灾难恢复是验证日志是否被成功归档的首要原因，但是还存在另外一个重要原因。如果日志没有归档的话，它们就会留在 LOGPATH 中。由于 LOGPATH 通常是在一个大小固定的文件系统中，如果日志文件没有归档，那么随着新日志的创建，文件系统就会慢慢地被填满。当出现这种情况时，DB2 将无法再创建日志文件，从而会停下来。

当调用 userexit 以归档日志文件时，将把信息写到两个地方。第一个地方是 userexit audit log，对于 userexit 收到的每个归档日志请求，都要写一个条目到这里。如果在 userexit 执行过程中发生了错误，那么还要将一条消息写入 userexit error log 文件中。这些日志文件位于 LOGPATH 中，文件名分别为 ARCHIVE.LOG 和 USEREXIT.ERR。

为了方便检查这些日志，可以编写一个简易脚本，为所有实例从这两个文件中捕捉最后 50 到 100 行(使用 tail 命令)，并通过电子邮件发送给自己。然后就可以在每天早晨将这些行与恢复历史信息放在一起研究。

DB2 会对整个数据库上发生过的备份情况进行记录，可以通过“ list history backup all for 数据库名”命令进行查看，具体如下所示：

```
db2  list history backup all for sample
匹配的文件条目数 = 1
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Op 对象 时间戳记+序列      类型  设备  最早日志      当前日志       备份标识
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 B  D  20080218191000001   F    D  S0000000.LOG S0000040.LOG
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 包含 5 表空间:
 00001 SYSCATSPACE
 00002 USERSPACE1
 00003 IBMDB2SAMPLEREL
 00004 DATA_SPACE
 00005 INDEX_SPACE
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    注释: DB2 BACKUP SAMPLE OFFLINE
开始时间: 20081218191000
结束时间: 20081218191012
```

```
    状态: A
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - -
EID: 8 位置: C:
```

除此之外，还要使用 db2ckbkp 检查备份文件的完整性，如下所示：

```
db2ckbkp -h SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
=====================
MEDIA HEADER REACHED:
=====================
       Server Database Name          -- SAMPLE
       Server Database Alias         -- SAMPLE
       Client Database Alias         -- SAMPLE
       Timestamp                   -- 20081119232532
       Database Partition Number      -- 0
       Instance                    -- DB2
       Sequence Number              -- 1
       Release ID                  -- C00
       Database Seed                -- 491AA028
       DB Comment's Codepage (Volume) -- 0
       DB Comment (Volume)           --
       DB Comment's Codepage (System) -- 0
       DB Comment (System)           --
       Authentication Value          -- 255
       Backup Mode                  -- 0
       Includes Logs                -- 0
       Compression                  -- 0
       Backup Type                  -- 0
       Backup Gran.                 -- 0
       Status Flags                 -- 11
       System Cats inc              -- 1
       Catalog Partition Number      -- 0
       DB Codeset                   -- UTF-8
       DB Territory                 --
       LogID                       -- 1221200288
       LogPath                     -- \DB2\NODE0000\SQL00001\SQLOGDIR\
       Backup Buffer Size            -- 3411968
       Number of Sessions            -- 1
       Platform                    -- 5
 The proper image file name would be:
SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
[1] Buffers processed:
############################################################
Image Verification Complete - successful.
```

11.7.5　维护实例目录和数据库目录的权限

在实例和数据库创建好以后，操作系统中会有相应的 DB2 用户和组。如果这些用户和组被删除后，将使 DB2 无法正常使用。另外，在实例目录下执行“chown(chmod) – R”命令会造成在服务器上无法连接数据库。如果遇到这种情况，正确的做法是把删除的用户和组添加回来，如果更改了权限，还要把权限改回来。

但是如果使用 chown(chmod) – R 或 chmod 777 更改了实例目录下的属组和权限，由于涉及的文件很多，如果不记得改了哪些权限，想改回来是十分困难的。建议在平时制定严格的维护规范，让所有的系统管理员和 DBA 都严格执行，特别是要告知系统管理员哪些目录下的文件不能删除，哪些目录下的文件权限是不能更改的。在实际维护的过程中，经常发生系统管理员在不知道危害的情况下进行了误操作。

11.7.6　查看磁盘空间

在使用归档日志模式的情况下，如果离线活动日志不能顺利归档到第三方路径，比如磁带机上，数据库容易出现磁盘满错误。另外，在使用 SMS 表空间模式的情况下，由于表空间所需实际空间是按需分配的，因此也有可能会出现磁盘满的问题。所以，DBA 每天需要查看活动日志目录所在的文件夹是否还有空间，以及 SMS 表空间对应的容器目录是否还有空间等。

关于数据库的活动日志目录可以使用 get db cfg 命令查看，找到活动日志目录后，可以使用操作系统命令(比如“df–k”等)查看该目录是否还有磁盘空间，注意一定不要手工删除活动日志。

对于 SMS 表空间对应的容器目录，也需要经常查看，防止 DB2 数据库管理器没有足够的空间来给 SMS 表空间分配新页。对于 DMS 表空间来说，由于空间是预分配的(在创建表空间的时候，空间已经提前分配)，因此通过查看磁盘的方式，不能正确得知其是否还有空余空间，而是应该使用 list tablespaces show detail 命令进行查看。比如对于表空间 USERSPACE1 来说，总计页数为 8192，总可用页数为 8128，已用页数为 3648，还剩可用页数为 4480，具体如下所示：

```
db2 list tablespaces show detail
表空间标识  = 4
 名称  = data_space
 类型 = 数据库管理空间
 内容  = 所有持久数据。大型表空间。
 状态  = 0x0000  详细解释:   正常
 总计页数  = 8192
 可用页数  = 8128
```

```
    已用页数  = 3648
    可用页数 = 4480
    高水位标记(页)  = 3648
    页大小(以字节计)  = 4096
    扩展数据块大小(页)  = 32
    预取大小(页)  = 32
```

命令成功完成。为了在单分区上查看 DMS 表空间是否还有可用页，可以使用“db2 list tablespaces show detail”。

用这种方式查看表空间使用情况时，可读性不是特别好。还可以利用类似下面这样的 SQL 脚本，输出类似 df –k，但可读性比较好：

```
   db2 select
substr(tbsp_name,1,18),tbsp_type,tbsp_free_size_kb,tbsp_utilization_percent
from sysibmadm.tbsp_utilization
   ---------------------------------------------------------------------
   表空间名称      TBSP_TYPE  TBSP_FREE_SIZE_KB    TBSP_UTILIZATION_PERCENT
   ------------------ ---------- -------------------- ----------------------
   SYSCATSPACE        DMS                      5824                   91.10
   TEMPSPACE1         SMS                         0                  100.00
   USERSPACE1         DMS                     17920                   44.88
   IBMDB2SAMPLEREL    DMS                     18688                   42.51
   SYSTOOLSPACE       DMS                     31872                    2.63
   SYSTOOLSTMPSPACE   SMS                         0                  100.00
   TS2                DMS                     31744                    2.36
     7 条记录已选择
```

11.8 数据库监控

为什么需要监控数据库呢？这可以有很多理由，其中最主要的就是为了确保系统当前或者在将来一段时间内不存在问题。在问题还未发生之前就检测到问题，并采取行动，总比等到问题已经发生而被动地做出反应要好。如果按照下面描述的那样监视 DB2 数据库系统，就可以在很多问题发生之前检测到它们，并维护系统的性能。

11.8.1 监控工具

通常，需要将对 DB2 的监控与对操作系统的监控结合起来，以便得到对数据库服务器上所发生一切的完整描述。单凭 DB2 工具一般不能提供完整的描述。

在捕捉用于分析的信息时，应确保同时捕捉 DB2 和操作系统的信息，因为我们不能把

在不同时间捕捉到的信息相关联。

1. Linux 和 UNIX 工具

在监控系统时，每过一段时间就拍一次快照。如果快照时间的跨度仅仅是一两分钟，那么就无法反映系统活动的实际状况。建议每隔一两分钟就拍一次，并持续至少一个小时。例如，为了捕捉 CPU、内存以及其他操作系统的使用情况，可以使用工具 vmstat、sar、topas、nmon、top 或 glance。

例如，vmstat 命令的参数如下：

- 参数 1：时间间隔，以秒计，该工具按照这个时间间隔捕捉系统信息。
- 参数 2：捕捉系统信息的次数。

为了连续一个小时每一分钟捕捉一次 vmstat 信息，并将输出写到名为 vmstat.out 的文件中，可以使用如下命令：

```
vmstat 60 60 > vmstat.out
```

为了连续一个小时每两分钟捕捉一次 vmstat 信息，并将输出写到名为 vmstat.out 的文件中，可以使用如下命令：

```
vmstat 120 30 > vmstat.out
```

既要捕捉正常/平均工作负载，也要捕捉峰值工作负载。虽然确保高效地处理正常工作负载很重要，然而同样重要的是，还应确保系统能够在不使服务器超载的情况下处理峰值工作负载。

2. DB2 工具

DB2 有很多工具可用于监控数据库和实例的活动，这些工具包括：

- Snapshot Monitors/SQL Snapshot 函数
- Event Monitor
- SYSIBMADM 动态性能视图

还有其他一些工具和日志也可以提供关于数据库和实例的信息，包括：

- 通知日志：在 Linux 和 UNIX 中，这是一个独立的文件；而在 Windows 中，这个文件被合并到 Event Log 中。
- DB2DIAG.LOG 可以使用 db2diag 命令过滤查看。
- Memory Visualizer。

11.8.2　计算数据库的大小

通过执行存储过程 GET_DBSIZE_INFO，可以计算出数据库的大小和最大容量：

```
    CALL GET_DBSIZE_INFO(?, ?, ?, 0)
    The procedure returns:
    Value of output parameters
    --------------------------
    Parameter Name  : SNAPSHOTTIMESTAMP
    Parameter Value : 2004-02-29-18.33.34.561000
    Parameter Name  : DATABASESIZE
    Parameter Value : 22302720
    Parameter Name  : DATABASECAPACITY
    Parameter Value : 4684859392
     Return Status = 0

    输出分析：
    DATABASESIZE 即是数据库的大小，单位为字节
    计算公式为：sum (used pages * page size) for each table space (SMS & DMS)

    DATABASECAPACITY 即是数据库的最大容量，单位为 bytes
    计算公式为：SUM (DMS usable_pages * page size) + SUM (SMS container size
+ file system free size per container).
```

注意：

如果多个 SMS 表空间容器被定义在同一文件系统中，那么文件系统的空闲空间只被计算一次。

11.8.3　监控表的物理大小

```
    select fpages from syscat.tables where tabname='T1'
```

注意：

这个结果的准确程度依赖于统计信息收集的及时与否。在用此语句监控表的物理大小之前，建议做一次 RUNSTATS。此结果并不包含 LONG、LOB 数据的大小。

11.8.4　监控单个索引的大小

```
    select substr(indschema,1,6) ischema,substr(indname,1,20)
iname,iid,substr(tabschema,1,6) tschema,substr(tabname,1,20)
tname,compress attr c attr,index compressed
i compressed,decimal(index object l size/1024,6,0)
lsz mb,decimal(index object p size/1024,6,0) psz mb,index requires rebuild
rebuild,large rids from
table(admin_get_index_info('','SAPBGP','/BIC/B0002788000')) AS X
```

注意：
此语句在 DB2 V9.7 之前的版本中不适用。

11.8.5　监控数据库实用工具的进度

```
db2 list utilities show detail
```

此语句可以监控如下实用工具的进度：

- LOAD
- BACKUP
- RESTORE
- RUNSTATS

11.8.6　监控数据库 crash recovery 进度

```
db2pd -db <dbname> -recovery
```

11.8.7　监控 catalog cache 命中率

```
    select
decimal((1-float(cat cache inserts)/float(cat cache lookups+1))*100,6,2)
catalog cache
    from sysibmadm.snapdb
```

11.8.8　监控 package cache 命中率

```
    select
decimal((1-float(pkg cache inserts)/float(pkg cache lookups+1))*100,6,2)
from sysibmadm.snapdb
```

11.8.9　监控排序溢出率

```
    select decimal(((sort overflows*1.0)/(total sorts*1.0+1))*100,6,2)
from sysibmadm.snapdb
```

11.8.10　监控正在 REORG 的表

```
    select * from sysibmadm.SNAPTAB_REORG
```

11.8.11　监控缓冲池命中率

可以执行如下 SQL 来监控缓冲池的命中率，一般建议缓冲池的命中率在 95%以上：

```
db2 "select substr(bp name,1,12) total hit ratio percent,
data hit ratio percent,index hit ratio percent from sysibmadm.
bp hitratio"
----------------------------------------------------------------------
```

```
TOTAL_HIT_RATIO_PERCENT DATA_HIT_RATIO_PERCENT INDEX_HIT_RATIO_PERCENT
----------------------- ---------------------- -----------------------
IBMDEFAULTBP                          62.38                  55.88
IBMSYSTEMBP4                              -                      -
IBMSYSTEMBP8                              -                      -
IBMSYSTEMBP1                              -                      -
IBMSYSTEMBP3                              -                      -
  5 条记录已选择。
```

11.8.12　监控高成本应用程序

可以使用性能管理视图来监控高成本应用程序。APPL_PERFORMANCE 视图可以帮助我们监控可能正在执行大型表扫描操作的应用程序：

```
select AGENT_ID, PERCENT_ROWS_SELECTED from sysibmadm.APPL_PERFORMANCE
```

PERCENT_ROWS_SELECTED 表示的是返回给应用程序的行数占从磁盘读取的行数的百分比。如果此值很低，那么表示该应用程序可能正在执行表扫描操作，通过创建索引可以避免应用程序执行该操作。使用此视图来标识可能有问题的查询，然后可以进行进一步调查，即查看 SQL 执行计划以确定是否能够减少查询在执行时读取的行数。

11.8.13　监控正在执行的时间最长的 SQL 语句

可以使用 LONG_RUNNING_SQL 管理视图来监控当前正在执行的运行时间最长的查询：

```
    select ELAPSED_TIME_MIN, APPL_STATUS, AGENT_ID from
sysibmadm.long_running_sql order by ELAPSED_TIME_MIN desc fetch first 5 rows only
```

通过使用此视图，可以监控那些查询已运行的时间长度以及这些查询的状态。如果某个查询已执行了很长时间并且正在等待锁，那么还可以使用对特定代理程序标识执行查询的 LOCKWAITS 或 LOCK_HELD 管理视图来进行进一步调查。LONG_RUNNING_SQL 视图还会指出正在执行的语句并允许标识可能有问题的 SQL。

11.8.14　监控 SQL 准备和预编译时间最长的 SQL 语句

可以使用 QUERY_PREP_COST 对已确定有问题的查询进行故障诊断。此视图可以指出查询的运行频率以及这些查询中每个查询的平均执行时间：

```
    select NUM_EXECUTIONS, AVERAGE_EXECUTION_TIME_S, PREP_TIME_PERCENT from
sysibmadm.QUERY_PREP_COST order by NUM_EXECUTIONS desc;
```

PREP_TIME_PERCENT 值向我们指出准备查询时耗用的时间在查询执行时间中所占

的百分比。如果编译和优化查询时耗用的时间几乎与查询的执行时间一样长，那么可以建议查询的所有者更改用于查询的优化类。降低优化类可以使查询更快地完成优化，从而更快地返回结果。但是，如果某个查询需要相当长的时间来进行准备，但要执行数千次(而不必再次进行准备)，那么更改优化类并不能提高查询性能。

11.8.15　监控执行次数最多的 SQL 语句

```
    select * from sysibmadm.TOP_DYNAMIC_SQL order by NUM_EXECUTIONS desc fetch
first 5 rows only
```

11.8.16　监控执行时间最长的 SQL 语句

```
    select * from sysibmadm.TOP_DYNAMIC_SQL order by AVERAGE_EXECUTION_TIME_S
desc fetch first 5 rows only
```

11.8.17　监控排序次数最多的 SQL 语句

```
    select STMT_SORTS, SORTS_PER_EXECUTION, substr(STMT_TEXT,1,60) as
STMT_TEXT from sysibmadm.TOP_DYNAMIC_SQL order by STMT_SORTS desc fetch first
5 rows only
```

11.8.18　监控引起锁等待的 SQL 语句

```
    select AGENT_ID,substr(STMT_TEXT,1,100) as statement,STMT_ELAPSED_TIME_MS
from sysibmadm.snapstmt where AGENT_ID in (select AGENT_ID_HOLDING_LK from
sysibmadm.snaplockwait  order by  LOCK_WAIT_START_TIME ASC FETCH FIRST 20 ROWS
ONLY ) order by STMT_ELAPSED_TIME_MS DESC
```

11.8.19　查找新创建的对象

必须关注是否有人在生产数据库中创建了新的对象，例如表、索引、存储过程等，这一点非常重要。新对象通常表明服务器上安装了新的应用程序，任何新的应用程序和/或对象都将影响系统的操作特征(operational characteristics)。

此外，新的对象将消耗数据库里的空间，因此重要的是在这些对象变得太大并可能填满一个表空间之前，将它们识别出来。如果这些对象不是由 DBA 创建的，那么很可能就是在错误的表空间中创建的，这样就会导致空间和/或性能问题。

这里有一些方法可用于检查系统中的任何新对象：

- 每周运行 db2look 并写报告到文件中，检查新输出与上周输出之间的不同。
- 从 SYSCAT.TABLES、SYSCAT.INDEXES 和 SYSCAT.PROCEDURES 中选择对象名称并检查新输出与上周输出之间的不同。在这些系统编目表中有个 create_time 字段，表示对象创建的时间。

对于任何不同之处，可以从编目表中判定该对象的 CREATOR，并利用该信息追溯到创建该对象的人。

11.8.20 查找无效对象

通过下面的 SQL 语句查找数据库中无效的数据库对象：

```
--查找无效的表
select tabname from syscat.tables where status='X'

--查找无效的触发器
select trigname from syscat.triggers where VALID='N'

--查找无效的视图
select viewname from syscat.views where valid='N'

--查找无效的视图
select routinename from syscat.routines where valid<>'Y' and valid is not null
```

11.8.21 检查表空间状态

使用 db2pd 或 db2 list tablespaces show detail 检查表空间状态，状态信息如图 11-4 所示。

十六进制值	十进制值	State
0x0	0	正常（请参阅 sqlutil.h 中的定义 SQLB_NORMAL）
0x1	1	停顿：SHARE
0x2	2	停顿：UPDATE
0x4	4	停顿：EXCLUSIVE
0x8	8	装入暂挂
0x10	16	删除暂挂
0x20	32	备份暂挂
0x40	64	正在前滚
0x80	128	前滚暂挂
0x100	256	复原暂挂
0x100	256	恢复暂挂（未使用）
0x200	512	禁用暂挂
0x400	1024	正在重组
0x800	2048	正在备份
0x1000	4096	必须定义存储器
0x2000	8192	正在复原
0x4000	16384	脱机并且不可访问
0x8000	32768	删除暂挂
0x10000	65536	不允许写入
0x20000	131072	正在装入
0x40000	262144	正在重新分发
0x80000	524288	正在移动
0x2000000	33554432	可以定义存储器
0x4000000	67108864	存储器定义处于“最终”状态
0x8000000	134217728	在前滚之前已更改存储器定义
0x10000000	268435456	DMS 重新平衡程序处于活动状态
0x20000000	536870912	正在进行 TBS 删除
0x40000000	1073741824	正在进行 TBS 创建

图 11-4 表空间状态信息

11.8.22　检查表状态

通过系统编目表 syscat.tables 中的 status 列来检查表的状态：

```
Select status from syscat.tables where tabname='T1'
```

状态信息如下：

```
C = set integrity pending
N = normal
X = inoperative
```

11.8.23　查找需要 REORG 的表和索引

当插入、更新和删除表中的行时，都要对表中的数据进行 REORG(重组)，以便：

- 按照索引的重要顺序重新群集(re-cluster)数据。
- 消除表和索引碎片。
- 去掉溢出(overflow)的记录。

REORGCHK 工具将对表进行检查，并表明需要对哪些表进行 REORG。可以对单个的表、所有用户表、某个特定模式中的所有表或所有系统编目表运行 REORGCHK 工具。还可以指示该工具是应该使用当前统计信息作为基础，还是应该首先收集新的统计信息。

为了对所有表运行 REORGCHK 工具，并确保正在使用当前统计信息，可使用命令：

```
reorgchk update statistics on table user
```

这里应将该命令的输出重定向到文件中以供进一步分析。

当查看 REORGCHK 工具的输出时，找到用于表的 F1、F2 和 F3 这几列，以及用于索引的 F4、F5、F6、F7 和 F8 这几列。如果这些列中的任何一列有星号(*)，就说明当前的表和/或索引超出了阈值。

记住，对于表来说，如果任何列中有一个星号，那么通常就需要 REORG 表。然而，由于很多表都拥有不止一个索引，按照定义，如果某个索引是 100%群集的，那么其他索引就不是群集的。因此，在判断是否 REORG 索引时，需要调查 REORGCHK 输出的索引部分，并考虑表上的所有索引。

对 REORGCHK 所使用的度量的考虑因素包括：

F1：属于溢出记录的行所占的百分比。当这个百分比大于 5%时，在输出的 F1 列中将有一个星号(*)。

F2：数据页中使用了的空间所占的百分比。当这个百分比小于 70%时，在输出的 F2 列上将有一个星号(*)。

F3：其中含有包含某些记录的数据的页所占的百分比。当这个百分比小于 80%时，在输出的 F3 列上将有一个星号(*)。

F4：群集率，即表中与索引具有相同顺序的行所占的百分比。当这个百分比小于 80%时，那么在输出的 F4 列上将有一个星号(*)。

F5：在每个索引页上用于索引键的空间所占的百分比。当这个百分比小于 50%时，在输出的 F5 列上将有一个星号(*)。

F6：可以存储在每个索引级的键的数目。当这个数字小于 100 时，在输出的 F6 列上将有一个星号(*)。

F7：在页中被标记为 deleted 的记录 ID(键)所占的百分比。当这个百分比大于 20%时，在输出的 F7 列上将有一个星号(*)。

F8：索引中空叶子页所占的百分比。当这个百分比大于 20%时，在输出的 F8 列上将有一个星号(*)。

在重组表的时候，可以选择指定 DB2 应该按哪个索引集群数据。例如，为了基于 ORGX 索引重组表 ORG，可以使用命令：

```
reorg table org index orgx
```

11.8.24　查找需要 RUNSTATS 的表和索引

DB2 优化器使用数据库统计信息来决定 SQL 语句的最佳执行计划。如果对表做了大量更新，那么应使用 RUNSTATS 命令采集新的数据库统计信息，并将这些信息存储在系统编目中。还应确保采集了任何新的表或索引的数据库统计信息。

为了捕捉表及其索引的统计信息，使用命令:

```
runstats on table <schema>.<tabname> with distribution and detailed indexes all
```

注意：

在使用 RUNSTATS 命令的时候，必须指定表的模式名。

可以使用如下语句来检查任何没有统计信息的表和索引：

```
select tabname from syscat.tables where stats_time is null
select indname from syscat.indexes where stats_time is null
```

可以使用如下语句来检查任何没有统计信息的索引：

```
select indname from syscat.indexes where stats_time is null
```

可以使用如下语句来查找超过 30 天没有收集统计信息的表和索引：

```
    select tabname from syscat.tables where stats_time < current timestamp -
30 days
select indname from syscat.indexes where stats_time < current timestamp - 30
days
```

11.8.25　定期清理 db2diag.log 文件

定期清理诊断日志是一个很好的习惯，这时日志要小得多，同时也容易编辑。这样一来，当数据库出现故障时，就不必回顾日志中过去的无用信息。在清除文件之前，应先做备份，以防在将来某个时候想要回头检查系统在某个时间点上曾发生过什么。

在 Windows 上，可以在 Event Viewer 中将事件日志保存到另一个文件中，方法是选择 Action 菜单，再选择 Save Log File As &选项。然后，通过选择 Action 菜单，再选择 Clear All Events 选项将条目从日志中清除。

注意：
用当前日期命名该文件是一个好的习惯，这使得在以后某天回头查看文件时更方便。

对于 Linux 和 UNIX 上的 db2diag.log 文件以及 administration notification log 文件，应该进行压缩，然后在命名时也使用当前日期。

在 Linux 或 UNIX 上，可以将*.nfy 和 db2diag.log 文件归档到一起，然后使用 gzip 或 compress 减少最终文件的大小。

11.8.26　查找异常增长的表空间和表

检查表和表空间，看看上个月它们的增长情况。如果知道表和表空间的增长速度，以及还剩下多少可用空间，就可以判断它们的增长趋势，避免潜在的异常空间增长问题。

通过使用以下语句，可以获得表空间的大小和可用空间的大小：

```
    Select TBSP_NAME, TBSP_USED_PAGES, TBSP_FREE_PAGES  from
sysibmadm.SNAPTBSP_PART
```

通过查看系统编目表，可以知道每个表的大小。只要统计信息是最新的，就可以准确无误。为了获得表的大小，可以使用语句：

```
    select tabname, npages from syscat.tables where tabname not like 'SYS%'
```

注意：
如果没有捕捉到某个表的统计信息，npages 上的值就是 NULL。

创建历史表来存储该信息，这样就可以详细跟踪表和表空间的空间使用情况。

11.8.27 数据库维护总结

当数据库运行了较长时间之后，随着数据的变化或增长，数据库中的应用会变得越来越慢，因而不得不对数据库进行统计信息更新、碎片整理和重新绑定工作以提高应用的性能，而且通常会一起考虑执行。处理流程如图 11-5 所示。

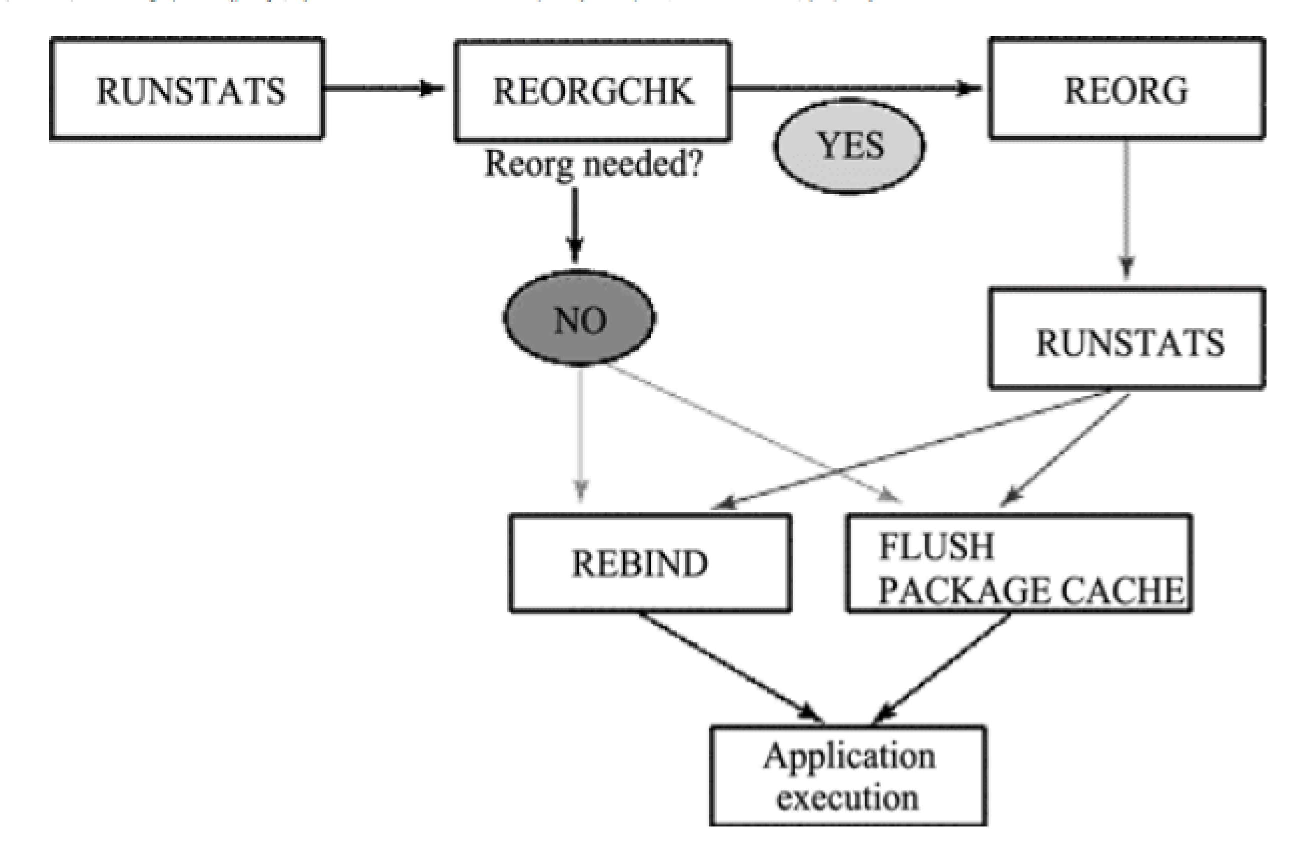


图 11-5 统计信息更新和碎片整理的处理流程

首先使用统计信息更新工具 RUNSTATS 更新相关数据库对象的统计信息，然后使用数据库的重组工具 REORG 对数据库的碎片进行这里，接着再使用 RUNSTATS 工具对数据库的统计信息进行更新，最后将数据库中相关的程序包重新绑定。这样就完成了完整的统计更新和碎片整理过程。在经过一定的时间周期后(一周、一个月或其他时间)，再次完成上面描述的完整过程。这样在数据库中访问数据时，就不会由于统计信息误差过大而选择成本高昂的访问策略。

那么对于上述循环过程的执行频率应该是何种频率呢？答案是：不一定。这需要根据实际情况来确定。最重要的就是要明确上述过程的最终目标是什么？上述过程的最终目标就是将数据库中数据的变化反映出来并让数据库做出相应的调整。所以数据库中数据变化得越快，上述过程的频率就应该越高；相反，数据库中数据变化越慢，上述过程的频率也就越低。如果数据库中的数据没有变化，甚至不需要执行上述过程。还有个需要关注的因素就是时间窗口，实际的应用程序是否能够为我们执行上述过程留出足够的时间窗口。比如 5×24 的应用，那么平时是没有时间窗口让我们来完成上述过程的，这些工作只能放到周末去执行。还有，假设有一个非常大的表，如果通过常规的方法——碎片检查、表和索引碎片整理，最后统计信息更新，那么可能要经过很长的时间。如果不是 7×24 的应用，建议最快的方法是把表中的数据导出(export)，然后用带 statistics 选项的 load replace 操作

来转载数据。这样不但做了表和索引的碎片整理，同时还做了统计信息更新，而且时间比常规方式会大大缩短。总而言之，最终采用何种频率、何种方式来完成上述过程，需要考虑上面的两方面因素来做决定。



数据库常用工具

DB2 数据库为我们提供了很多功能强大的工具，这些工具能够帮助我们解决某一特定的问题。本章主要讲解 DB2 中一些最常用的工具。

本章主要讲解如下内容：

- 解释工具
- 索引设计工具
- 基准测试工具
- 数据一致性检查工具
- db2look 工具
- 其他工具

12.1　解释工具

实际运行中，应用中的某些 SQL 语句往往比较消耗资源，当我们使用监控工具定位特定的 SQL 语句时，我们需要对该 SQL 语句做解释分析，这就需要用到解释工具。DB2 提供了 Visual Explain、db2expln、dynexpln 和 db2exfmt 等几种常用工具。

12.1.1　Visual Explain(可视化解释)

Visual Explain 是一种 GUI 工具，它为数据库管理员和应用程序开发人员提供了查看为特定 SQL 语句选择的访问计划的图形化表示的能力。但 Visual Explain 只能用于查看解释快照数据或人工输入 SQL 或脚本，要查看已收集并写入了解释表的全面解释数据，就必须使用 db2exfmt 或 db2expln 工具。

1. timeron

为了分析解释信息，您需要了解的最重要的事情就是 timeron 概念。timeron 是 DB2 优

化器使用的一种成本度量单位，用于计算查询完全执行所需的时间和资源数量。timeron 是时间、CPU 占用率、磁盘 I/O 和其他一些因素的综合。由于这些参数的值不断变化，执行查询所需的 timeron 数量也是动态的，每次执行都有所不同。

timeron 也是一种创造出来的度量单位，因此没有什么公式可以将执行查询所需的 timeron 数转换成秒数。除此之外，timeron 可以帮助您确定一种查询执行途径是否比另一种更快(如果执行查询所需的 timeron 数在两次编译之间相差 10 或 20，那么不必担心，因为这可能仅仅是由于 CPU 活动、磁盘活动或数据库使用情况发生了变化造成的)。

2. SQL 编译

在数据库中执行任何 SQL 语句之前，必须首先准备 SQL 语句。在此过程中，SQL 语句会被简化为经过优化器查询重写后的优化的 SQL 语句，DB2 优化器随后可对此语句进行分析。这条优化后的 SQL 语句就是所谓的 SQL 语句的存取计划(access plan)，在整个优化过程中发挥作用。图 12-1 展示了 SQL 语句的编译过程。

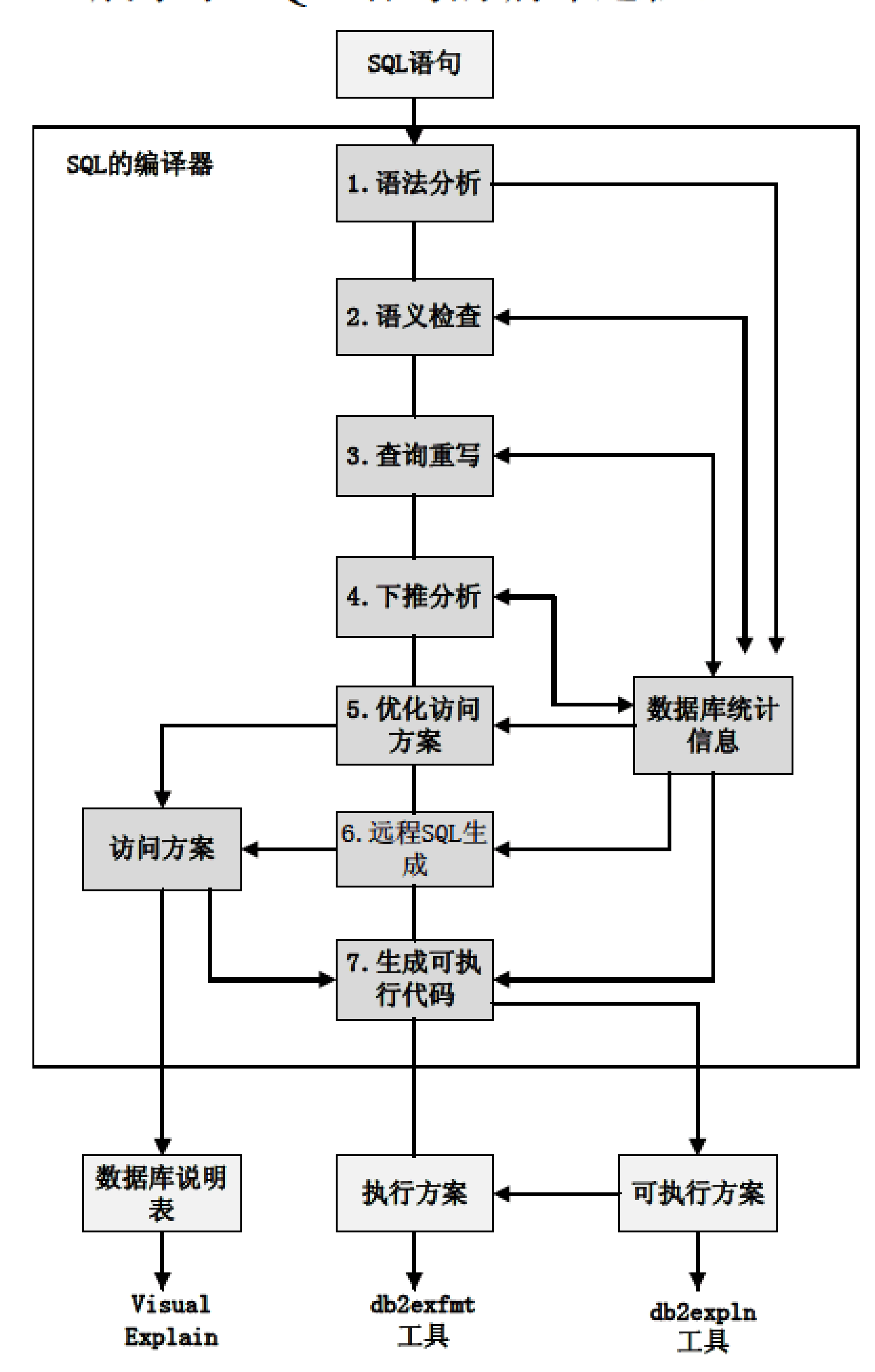


图 12-1　SQL 编译过程

SQL 编译过程的最终输出是访问计划。访问计划是 DB2 用于执行 SQL 语句的路径和步骤。这是由所有可用解释工具显示的信息。乍看上去，访问计划似乎非常复杂。但在具备了一定经验之后，您会很快发现它们实际上非常容易阅读和分析。

3. 激活 Visual Explain

只要收集到了全面解释(explain all)和(或)解释快照(explain snapshot)数据，关于数据收集实现和方法的信息就会记录在 EXPLAIN_INSTANCE 解释表中。你可随时通过“说明语句历史记录”(Explained Statement History)窗口查看此信息。要激活 Explained Statement History 窗口，请在 Control Center 中高亮显示适当的数据库，并在 Control Center 菜单中选择 **Selected > Show Explained Statement History** 即可。图 12-2 展示了在为一条 SQL 语句收集了解释快照数据之后，Explained Statement History 窗口的外观。

注意：

本例中使用的 DB2 环境是中文环境，如果读者的数据库环境是英文，只需对照相应术语的中英文翻译即可。



图 12-2　“说明语句历史记录”窗口

一旦打开“说明语句历史记录”窗口，就可以使用 Visual Explain 来分析解释的快照数据。显示这些数据的方法是：高亮显示一条记录，并在窗口的主菜单中选择 **Statement (语句)> Show Access Plan(显示访问计划)**。图 12-3 展示了以这种方法为以下查询创建的 Access Plan Gragh 窗口(此查询可在随 DB2 提供的 SAMPLE 数据库上执行)：

```
select name,balance from account where acct_id=47030
```

另一方面，还可为新查询收集解释快照数据，相应的访问计划可通过在“说明语句历史记录”窗口的主菜单中选择 **Statement > Explain Query...**显示出来。在选中这些菜单项时，Explain Query Statement 窗口将打开，并提示您为查询输入文本。图 12-4 展示了以一个简单的查询填充后的 Explain Query Statement 窗口。

Access Plan Graph 窗口中展示的访问计划的每个组件都可供单击，单击后即可看到关

于该组件的详细信息。例如，若选中图 12-3 所示的访问计划中的操作符，则 Operator Details 窗口中将显示如图 12-5 所示的详细信息。

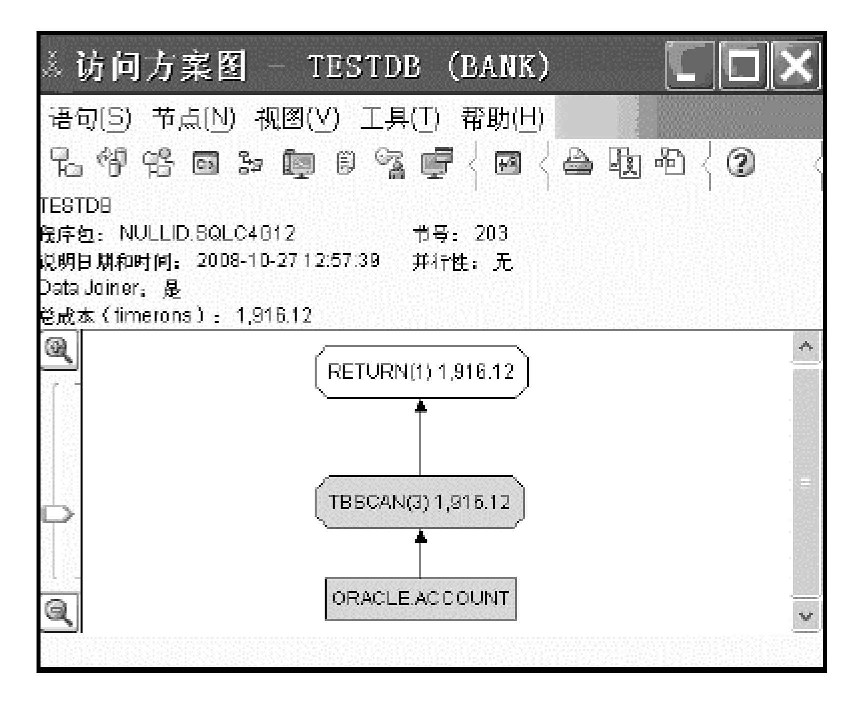


图 12-3　Access Plan Graph 窗口



图 12-4　Explain Query Statement 窗口



图 12-5　Operator Details 窗口

在分析访问计划以定位性能瓶颈时，最好尝试单击所有不同的对象类型，以便充分了解已有的查询信息。

4. Visual Explain 组件

您可能已经注意到，Access Plan Graph 窗口中提供的输出(参见图 12-3)由层次化图形构成，表示处理为指定查询选定的访问计划时所必需的不同组件。计划中的各组件都显示为一种称为节点的图形对象。存在两种类型的节点：

- **操作符(Operator)**。操作符节点用于确定是否必须在数据上执行一项活动，或者通过表或索引生成的输出。
- **操作对象(Operand)**。操作对象节点用于确定对其进行操作的实体(例如，表可以是表扫描操作符的操作对象)。

操作对象

典型情况下，操作对象节点用于确定表、索引和表队列(表队列用于使用了内部分区并行操作的情况)，它们在层次图中的符号分别是矩形(表)、菱形(索引)和平行四边形(表队列)。图 12-6 给出了表和索引操作对象的示例。

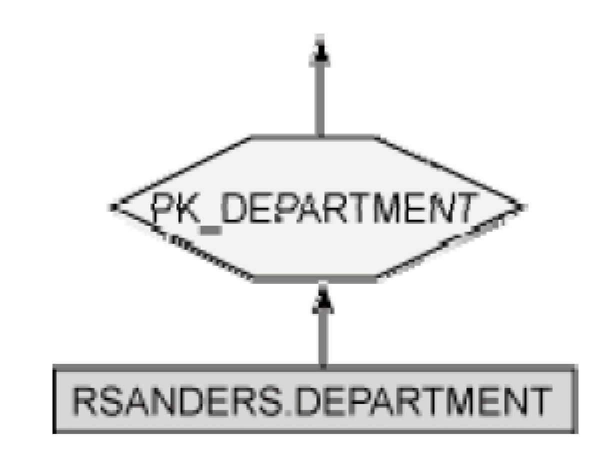


图 12-6　表和索引操作对象示例

操作符

另一方面，操作符节点用于确定从插入操作到索引或表扫描的一切活动。操作符节点在层次图中的符号为八边形，表示数据的访问方法、表的连接方法以及其他一些因素，例如是否要执行排序操作等。表 12-1 列出了访问计划层次图中较为常见的操作符。

表 12-1　常见 Visual Explain 操作符

操　作　符	所执行的操作
CMPEXP	计算表达式(仅用于调试模式)
DELETE	从表中删除行
EISCAN	扫描用户定义的索引，产生一系列简化的行

(续表)

操　作　符	所执行的操作
FETCH	使用指定的记录标识符从表中获取列
FILTER	通过应用一个或多个谓词过滤数据
GENROW	生成行表
GRPBY	按指定列或函数的公共值组织行，并对集合函数求值
HSJOIN	显示散列连接，其中一个或多个表在连接列上是混编的
INSERT	向表中插入行
IXAND	对两个或多个索引扫描得到的行标识符(RID)进行 AND 运算
IXSCAN	使用可选的启动/停止条件扫描表索引，产生有序的行流
MSJOIN	显示合并连接，其中外部表和内部表必须按连接谓词的顺序排列
NLJOIN	显示嵌套循环连接，为外部表中的各行访问内部表一次
PIPE	翻译行(仅用于调试模式)
RETURN	将查询返回的数据显示给用户
RIDSCN	扫描行标识符(RID)列表，该列表是从一个或多个索引中获得的
RPD	远程计划的操作符。与 DB2 V8 中的 SHIP 操作符极为类似(之前版本中的 RQUERY 操作符)，唯一的不同在于不包含 SQL 或 XQuery 语句
SHIP	从远程数据源中检索数据。在联合系统中使用
SORT	按特定类的顺序排序行，可以选择消除重复条目
TBSCAN	通过直接从数据页中读取所有数据而检索行
TEMP	将数据存储在临时表中以便读回(很可能要读回多次)
TQUEUE	在数据库代理之间传输表数据
UNION	串联来自多个表的行流
UNIQUE	消除特定列值重复的行
UPDATE	更新表中的行
XISCAN	扫描 XML 表的索引
XSCAN	在 XML 文档节点子树中导航
XANDOR	允许为多个 XML 索引应用 ANDed 和 ORed 谓词

图 12-7 展示了一些更为常见的操作符示例。在这些示例中，执行了 3 种不同的行动：两个表执行了表扫描，两个数据集使用散列连接算法连接。

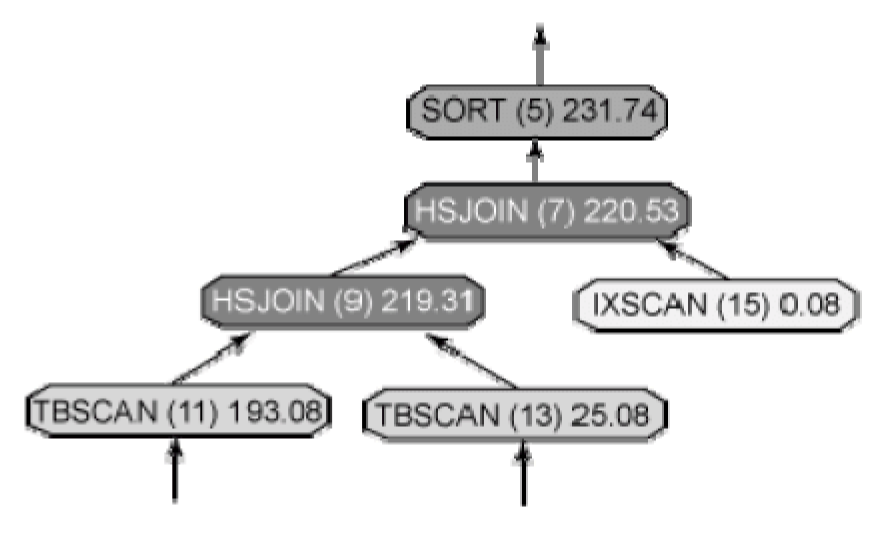


图 12-7 一些常见操作符

连接符和 RETRUN 操作符

箭头说明数据从一个节点流向另一个节点的方式，它将层次图中的所有节点连接在一起，RETURN 操作符通常用于终止这一过程。RETURN 操作符表示最终结果集产生，并包含关于查询的汇总信息以及所完成的 SQL 语句返回的内容。使用 RETURN 显示的 timeron 值表示按 timeron 度量的时间总长度，是完成查询所必需的时间。图 12-8 展示了 RETURN 操作符的一个示例。



图 12-8 RETURN 操作符

5. 影响查询性能的因素

数据库环境的配置参数、统计信息、I/O 设计和用于准备查询的查询优化级别对于查询的准备方式、执行方式有着重大的影响。

显示优化参数

Visual Explain 可迅速汇总影响查询编译的所有参数，并在一个汇总窗口中显示出来。这个窗口就称为 Optimization Parameters 窗口，通过在 Access Plan Graph 窗口的主菜单中选择 **Statement > Show Optimization Parameters** 可调用此窗口。图 12-9 展示了 Optimization Parameters 窗口在激活时的外观。

Optimization Parameters 窗口中包含的部分配置参数如下：

- **AVG_APPLS**(平均应用程序)：此参数表示为数据库并发运行的应用程序平均数量。DB2 使用此信息来确定排序空间和缓冲池使用得有多么频繁，并确定查询能够使用的空间有多少。

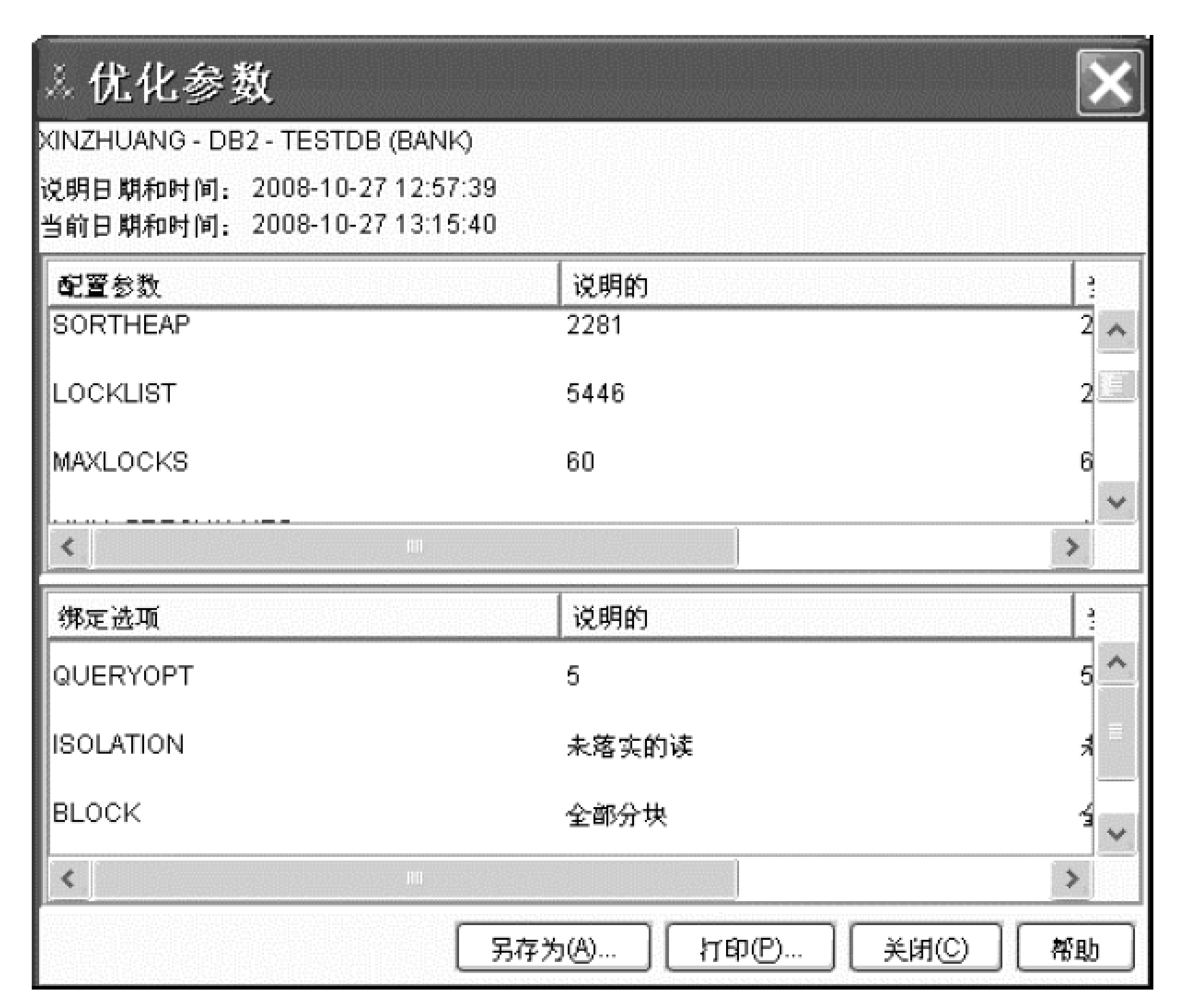


图 12-9 “优化参数”(Optimization Parameters)窗口

- **SORTHEAP**(排序堆)：排序堆是执行排序时可用的内存空间数量。若排序需要的内存多于排序堆中的可用内存，则部分排序数据将不得不分页到磁盘上(这会对性能造成严重的负面影响)。
- **LOCKLIST**(锁列表)：该参数表示 DB2 可用于存储各应用程序的锁定信息的内存数量。若锁列表空间过小，则 DB2 可能必须逐步升级(escalate)部分锁，以便为应用程序具有的所有锁腾出空间。
- **MAXLOCKS**(最大锁列表百分比)：该参数控制整个锁列表空间中有百分之多少的空间可为应用程序所有。若应用程序具有过多的锁，从而试图占用过多的内存，DB2 则升级部分锁以释放锁列表中的空间。
- **NUM_FREQVALUES**(频率值数)：DB2 RUNSTATS 实用工具使用频率值数来控制 DB2 将在内存中保留多少使用频率最高的值。优化器使用该信息来确定 WHERE 子句中的一个谓词将消耗结果集的多少百分比。
- **NUM_QUANTILES**(数据分位数)：DB2 RUNSTATS 实用工具使用分位数来控制为列数据捕获多少分位。增加分位数将给予 DB2 关于数据库中数据分布情况的更多信息。
- **DBHEAP**(数据库堆)：数据库堆控制数据库对象信息的可用内存量。对象包括索引、缓冲池和表空间。事件监控器和日志缓冲区信息也存储在这里。

● **CPUSPEED**(CPU 速度)：计算机的 CPU 速度。若此值设置为 –1，则 DB2 使用 CPU 速度度量程序来确定恰当的设置。
● **BUFFPAGE 和缓冲池大小：**优化器可在优化数据过程中使用的缓冲池大小。增加或减少缓冲池大小会对访问计划产生显著影响。

12.1.2　db2expln

在包含嵌入式 SQL 语句的源代码文件绑定到数据库时(无论是作为预编译流程的一部分，还是在延迟绑定过程中)，DB2 优化器将分析遇到的每一条静态 SQL 语句，并生成相应的访问计划，此访问计划随后以程序包的形式存储在数据库中(syscat.packages)。给定数据库名称、包名称、包创建者 ID、部分号(若指定了部分号为 0，则处理包的所有部分)，db2expln 工具即可为存储在数据库系统目录中的任何包解释并说明其访问计划。由于 db2expln 工具直接处理包而非全面解释数据或解释快照数据，因此通常用来获取那些已选定用于未捕获其解释数据的包的访问计划的相关信息。但由于 db2expln 工具仅可访问已存储在包中的信息，因此只能说明所选的最终访问计划的实现，不能提供特定 SQL 语句优化方式的信息。

若使用额外的输入参数，db2expln 工具则还可用于解释动态 SQL(不包含参数标记的动态 SQL 语句)语句(db2expln 的早期版本不支持动态 SQL(如 DB2 V7)，需要使用 dynexpln 工具，有了新版本后，可以不再使用 dynexpln，只用 db2expln 就可以了)。

对于数据库 SAMPLE 中的程序包 DB2INST1.P0203450，使用下面命令：

```
db2expln -d SAMPLE -g -c db2inst1 -p P0203450 -s 0 -t
```

其中：

● -g 是指给出存取计划的图形输出(用字符模拟)。
● -s 0 是指分析所有的 SQL 命令。

下面是相应的输出：

```
DB2 Universal Database Version 9.7, 5622-044 (c) Copyright IBM Corp. 1991, 2009
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL and XQUERY Explain Tool
DB2 Universal Database Version 9.7, 5622-044 (c) Copyright IBM Corp. 1991, 2009
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL and XQUERY Explain Tool
******************** PACKAGE ***************************************
Package Name = "db2ara"."P1518068452"  Version = ""
       Prep Date = 2012/09/16
       Prep Time = 18:32:49
       Bind Timestamp = 2012-09-16-18.32.49.070634
```

```
        Isolation Level         = Cursor Stability
        Blocking                = Block Unambiguous Cursors
        Query Optimization Class = 5
        Partition Parallel       = No
        Intra-Partition Parallel = No
        SQL Path                = "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM",
                                  "DB2ARA"
---------------------- SECTION ----------------------------------------
---------------------- SECTION ----------------------------------------
Section = 2
Statement:
  update POC set A=A+100  where A=5
Section Code Page = 1208
Estimated Cost = 21.288315
Estimated Cardinality = 1.792000
Access Table Name = DB2ARA.POC  ID = 5,45
|  #Columns = 0
|  Skip Inserted Rows
|  Skip Deleted Rows
|  Evaluate Block/Data Predicates Before Locking Row
|  May participate in Scan Sharing structures
|  Fast scan, for purposes of scan sharing management
|  Relation Scan---------全表扫描
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Exclusive
|  |  Row  : Update
|  Sargable Predicate(s)
|  |  #Predicates = 1
Update:  Table Name = DB2ARA.POC  ID = 5,45
End of section
Optimizer Plan:
       Rows
     Operator
       (ID)
       Cost
      1.792
       n/a
     UPDATE
      ( 2)
     21.2883
    /       \
  1.792    244
```

```
  n/a     n/a
 TBSCAN   Table:
  ( 3)    DB2ARA
 7.73383  POC
    |
  244
  n/a
 Table:
 DB2ARA
 POC
…
```

可以用下面的命令找出数据库中存在的程序包：

```
db2 "select pkgschema, pkgname from syscat.packages where pkgname =' P1518068452'"
```

上面例子中的程序包 DB2INST1.P0203450 是存储过程。

查看动态 SQL 的例子：

```
$db2expln -d sample -q "select * from employee" -t
DB2 Universal Database Version 9.7, 5622-044 (c) Copyright IBM  1991, 2009
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL and XQUERY Explain Tool
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL and XQUERY Explain Tool
******************** DYNAMIC ***************************************
==================== STATEMENT ==========================================
       Isolation Level          = Cursor Stability
       Blocking                 = Block Unambiguous Cursors
       Query Optimization Class = 5
       Partition Parallel       = No
       Intra-Partition Parallel = No
       SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM",
                                  "DB2ARA"
Statement:
   select *   from employee
Section Code Page = 1208
Estimated Cost = 7.597286
Estimated Cardinality = 42.000000
Access Table Name = DB2ARA.EMPLOYEE  ID = 2,6
|  #Columns = 14
|  Skip Inserted Rows
|  Avoid Locking Committed Data
|  Currently Committed for Cursor Stability
|  May participate in Scan Sharing structures
```

```
    |  Scan may start anywhere and wrap, for completion
    |  Fast scan, for purposes of scan sharing management
    |  Scan can be throttled in scan sharing management
    |  Relation Scan
    |  |  Prefetch: Eligible
    |  Lock Intents
    |  |  Table: Intent Share
    |  |  Row   : Next Key Share
    |  Sargable Predicate(s)
    |  |  Return Data to Application
    |  |  |  #Columns = 14
    Return Data Completion
    End of section
```

12.1.3　db2exfmt

与 db2expln 工具不同，db2exfmt 工具用于直接处理已收集并存储在解释表中的全面解释数据或解释快照数据。给定数据库名和其他限定信息，db2exfmt 工具将在解释表中查询信息、格式化结果，并生成一份基于文本的报告，此报告可直接显示在终端上或写入 ASCII 文件。

所有 explain 输出(包括 Visual Explain)都是从下往上读的，如图 12-10 所示。

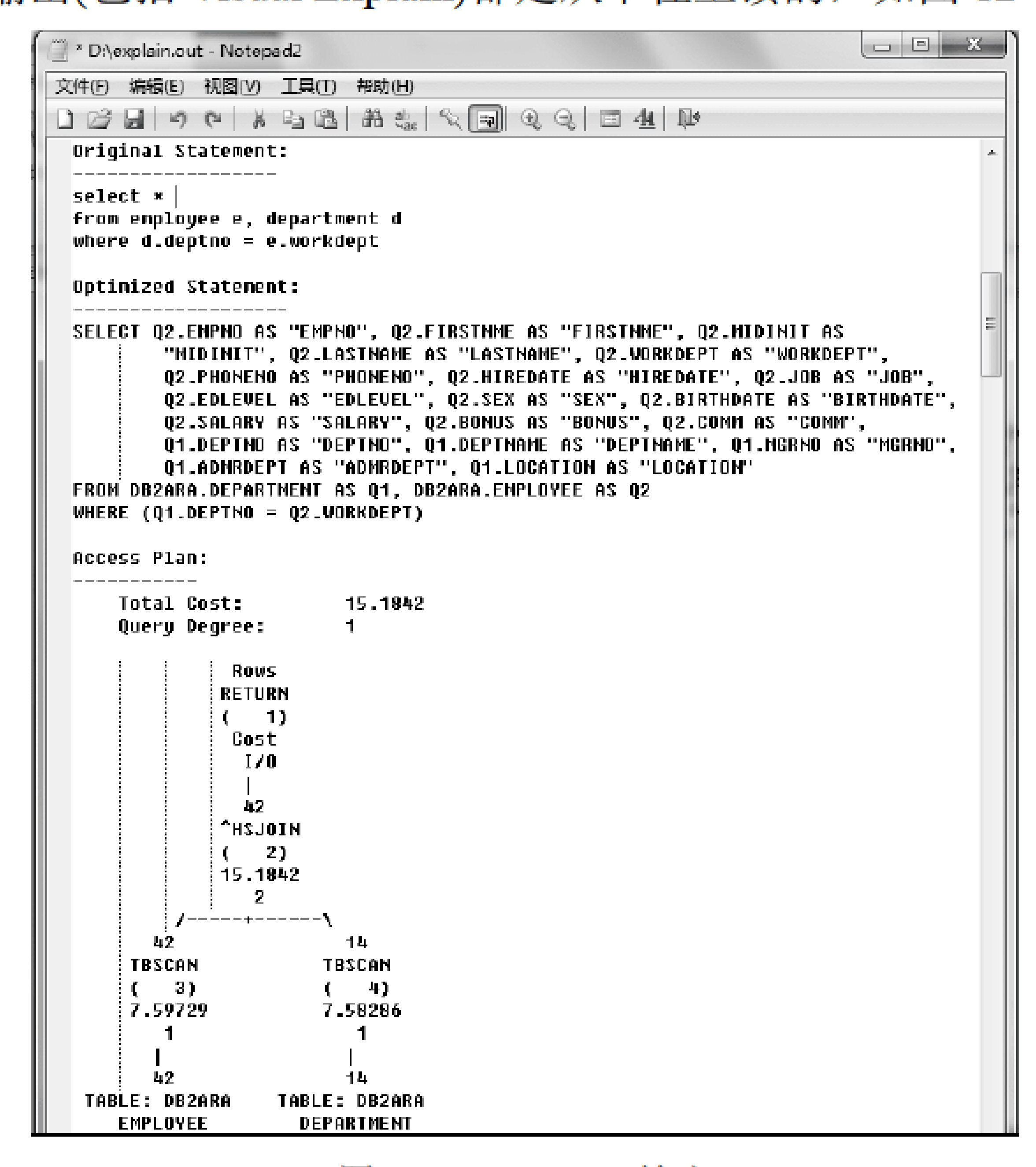


图 12-10　explain 输出

这里不像 Visual Explain 那样将所有细节显示在不止一个屏幕上，而是将所有细节列在输出文件中。图 12-10 中的每个操作符都编了号，当您往下查看时，每个操作符都将被详细解释。例如，图中的一个操作符可以作如下解释：

读 Text Explain 操作符

```
         42   - # of rows returned (based on statistics calculation)
        HSJOIN  - type of operator
        (   2) - operator #
        15.1842      - cumulative timerons
          2     - I/O costs
```

返回的行数、timeron(cost)数和 I/O 都是优化器估计的，在某些情况下可能与实际数字不符。timeron 的概念我们前面已经讲过，它是 DB2 的成本度量单元，用于给出对数据库服务器在执行同一查询的两种计划时所需的资源或成本的粗略估计。估计时计算的资源包括处理器和 I/O 的加权成本。

可以使用 db2exfmt 来解释单独一条语句，例如：

为一条语句生成 Text Explain 输出

```
explain all for
       SQL_statement
db2exfmt -d
       dbname -g tic -e
       explaintableschema -n % -s % -w -1 -# 0 -o
       outfile
```

如果要为用";"隔开的几条"explain all'语句构建文本文件，就可以一次解释多条语句。

为多条语句生成 Text Explain 输出

```
db2 -tf
       file with statements
db2exfmt -d
       dbname -g tic -e
       explaintableschema -n % -s % -w % -# 0 -o
       outfile
```

12.1.4　各种解释工具的比较

如你所见，可用于显示全面解释数据和解释快照数据的不同工具有着很大的差异，无论是在复杂性方面还是在功能方面。表 12-2 总结了几种可用工具，并强调了各自的特征。要使解释工具发挥出最好的效果，你应在选择工具时考虑环境和需求。

表 12-2　可用解释工具的比较

所 需 特 征	Visual Explain	db2exfmt	db2expln
用户界面	图形化	基于文本	基于文本
文本输出	否	是	是
来源 Explain Tables	是	是	否
快速但粗略的静态 SQL 分析	否	否	是
静态 SQL 支持	是	是	是
动态 SQL 支持	是	是	是
CLI 应用程序支持	是	是	否
详细的 DB2 优化器信息可用	是	是	否
适于分析多条 SQL 语句	否	是	是

12.1.5　如何从解释信息中获取有价值的建议

当分析 explain 的输出信息时，应该注意以下问题：

- 对相同的一组列和基本表使用的 ORDER BY、GROUP BY 或 DISTINCT 操作符将从索引或物化查询表(MQT)中受益，因为消除了排序。explain 用来帮助确保索引被正确地用于连接谓词、本地谓词以及 GROUP BY 和 ORDER BY 子句，以避免排序。
- 代价较高的操作，例如大型排序、排序溢出以及对表的大量使用，都可以受益于更多的排序空间(sortheap)、更好的索引、更新的统计信息或调优的 SQL。
- 表扫描也可以从索引中受益。
- 完全索引扫描或无选择性的索引扫描，其中不使用 start 和 stop 关键字，或者使用这两个关键字，但是有很宽的取值范围。这样的扫描性能会很差，尝试调整。
- 表的连接类型，利用您对表中数据的了解来确定正采用的连接类型是否正确，以及正在用于连接内部表和外部表的表是否正确。
- 未充分地利用索引。查询是否按您的希望使用了索引，应确保未在您理所当然地认为应该具有索引的表上进行表扫描。此问题可通过查看执行计划轻松应对。若确实存在索引，则检查基数或索引键的顺序。确保索引的集群度。
- 表基数和“SELECT *”的使用。有时，由于您要返回的列数，DB2 优化器会判定扫描整个表的速度更快。有可能表非常小，也有可能扫描索引并返回大量行(返回表中的所有列)的效率很低。尝试仅返回那些您确实需要的列。查看查询各部分返回的列，观察您是否确实需要这些列，并观察这是否是表扫描发生的原因。同样，考虑使用索引中包含的列。
- 优化级别的设置是否合适。

12.2 索引设计工具(db2advis)

12.2.1 DB2 Design Advisor(db2advis)

DB2 UDB V8.2 引入了一个名为 Design Advisor 的新工具，它对 DB2 早期版本中的 db2advis 工具做了更多扩展，提供了更强大的功能，该工具使用范围更广，可用来替代 db2advis，如图 12-11 所示。

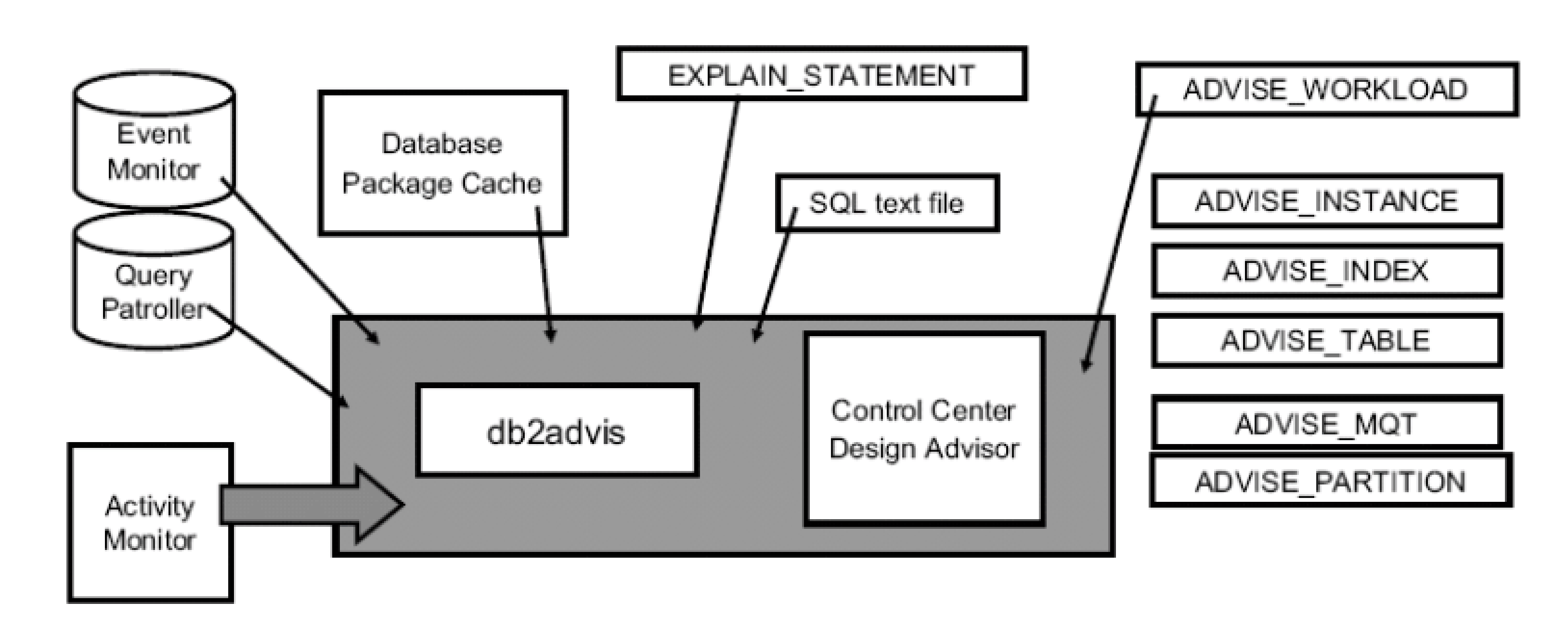


图 12-11 Design Advisor

Design Advisor 提供的建议能与数据库调优专家的建议相媲美。对于非专家来说，该工具的好处是可以获得更好的设计。对于专家来说，Design Advisor 可以节省他们宝贵的时间，因为 Design Advisor 可以提供初始设计，然后由专家进一步改进设计。Design Advisor 还可以提供对专家设计的独立确认。

要使用 Design Advisor，首先需要在相应数据库里执行 EXPLAIN.DDL 以存储各种数据。

```
cd $INSTHOME/sqllib/misc
db2 -tf EXPLAIN.DDL
```

使用 Design Advisor 的第一步是收集和描述提供给 Design Advisor 的工作负载。可以让 Design Advisor 从以下输入获取工作负载：

- 最近的 SQL 语句(来自动态 SQL 快照)。
- Query Patroller 语句(更适用于数据仓库数据库)。
- 静态 SQL 语句(来自应用程序包)。
- 解释后的 SQL 语句。

- Event Monitor 语句。
- 来自包含事务的文件。
- Activity Monitor 捕获的 SQL 语句。

我们可以为每个事务(每个 SQL)赋予 1、10、100 或 1000 的频率。这将导致 Design Advisor 对频率值为 10 的事务的重视程度是对频率值为 1 的事务的重视程度的 10 倍。

收集完事务并设置好每个事务的频率后，就可以运行命令 db2advis 以获取关于索引的建议，如下所示。如果创建了索引，SQL 执行成本可以提高 81.31%。

```
    db2advis -d SAMPLE -i sql1.txt -disklimit 90 -o sql1.out
    execution started at timestamp 2012-09-17-17.33.44.496579
    found [1] SQL statements from the input file
    Recommending indexes...
    total disk space needed for initial set [  10.493] MB
    total disk space constrained to         [  90.000] MB
    Trying variations of the solution set.
      1  indexes in current solution
     [43388.6641] timerons  (without recommendations)
     [8106.5694] timerons  (with current solution)
     [81.31%] improvement
    -- LIST OF RECOMMENDED INDEXES
    -- ===========================
    -- index[1],   10.493MB
       CREATE INDEX "DB2CMP"."IDX1209170933450" ON "DB2CMP"."T BATCHGATHERDTL"
       ("SERIAL NO" ASC, "TR ACDT" ASC, "BATCH NO" ASC) ALLOW
       REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;
       COMMIT WORK ;
    -- RECOMMENDED EXISTING INDEXES
    -- ============================
    -- RUNSTATS ON TABLE "DB2CMP"."T BATCHGATHERDTL" FOR SAMPLED DETAILED INDEX
"DB2CMP"."PK BATCHGATHERDTL" ;
    -- COMMIT WORK ;
```

12.2.2 DB2 Design Advisor(db2advis)案例讲解

现在我们将讲解一个案例，不过这次使用的是命令行而不是 GUI。

下面是这个案例中使用到的命令：

```
    db2advis -d sample -i top1.sql -m IMCP -k LOW  -l 700 -c TEMP_TBS -f
```

要点包括：

-m IMCP：规定 Design Advisor 应该考虑新的索引(I)、MQT(M)，将标准的表转换成 MDC 表(C)，并且重新为已有的表分区(P)。默认情况是只考虑索引。

-k LOW：规定将工作负载压缩到 low 级别。结果，Design Advisor 将分析您提供的更大一组的工作负载。默认情况是中等(medium)。

-l 700：规定任何新的索引、MQT 等都不能消耗多于 700MB 的空间。默认情况是数据库总体规模的 20%。

-c TEMP_TBS：规定使用表空间 TEMP_TBS 作为生成 MQT 建议的临时工作空间。如果您想要 MQT 建议，并且正在运行 DPF(多分区)示例，那么这个选项是必需的。否则，这个参数是可选的。

另一个有用的选项(没有给出)是-o *output_file* 。该选项保存脚本，以便在文件中创建建议的对象。

当该命令执行时，它描述正在进行的工作，下面显示了其中的一部分。至此，Design Advisor 已经生成了除 MDC 之外的所有有关对象的建议。

```
Cost of workload with all recommendations included [1306186] timerons
27 indexes in current solution
3 partitionings in current solution
8 MQTs in current solution
```

建议集有 27 个索引(新索引或已有的索引)、3 个分区(与 DPF 相关的更改，例如新的分区键或表空间)以及 8 个 MQT(新的或已有的)。

接下来，Design Advisor 分析 MDC，并在完成时显示以下信息：

```
3 clustering dimensions in current solution
[12305400] timerons(without any recommendations)
[1042873] timerons(with current solution)
[91.53%] improvement
```

“3 clustering dimensions”意味着 Design Advisor 建议 3 个 MDC 维。这 3 个 MDC 维可以同时在一个表上，也可以在不同的表上。例如，3 个维都在表 A 上，或者其中一个维在表 A 上，另外两个维在表 B 上。性能统计信息指的是所有建议的性能，而不仅仅是 MDC 建议的性能。“timerons(without any recommendations)”这一项指的是现有设计能取得的性能，而“with current solution”指的是实施这些建议后估计能取得的性能。

接着，Design Advisor 以 DDL 格式显示建议，并且该 DDL 已经被注释掉。这些建议以如下顺序出现：

- 包括 MDC 或分区建议的基本表。
- MQT 建议(首先是新的 MQT，然后是要保留的已有的 MQT，最后是未使用的 MQT)。
- 新的集群索引(如果有的话)。
- 索引建议(新的，保留的，然后是未使用的)。

关于更改表的建议如下所示：

```
-- CREATE TABLE "ORACLE"."LINEITEM"("L_ORDERKEY BIGINT NOT NULL,
-- "L_PART" INTEGER NOT NULL,
-- "L_SUPPKEY" INTEGER NOT NULL,
-- "L_LINENUMBER" INTEGER NOT NULL,
-- "L_SHIPINSTRUCT" CHAR(25)  NOT NULL,
(11 other columns omitted from this example)
-- MDC409022109290000 GENERATED ALWAYS AS(((INT(L_SHIPDATE))/7) )
-- ---- PARTITIONING KEY("L PARTKEY") USING HASHING
-- ---- IN "TPCDLADT"
-- ORGANIZE BY(
-- MDC409022109290000,
-- L SHIPINSTRUCT )
-- PARTITIONING KEY(L ORDERKEY) USING HASHING
-- IN TPCDLDAT
--;
-- COMMIT WORK ;
```

注意，这里建议新的分区键(L_ORDERKEY)，用以替代当前的分区键(L_PARTKEY)，后者被注释掉了。这个表的 MDC 建议(ORGANIZE BY 子句)包括两个维：生成的列(INT(L_SHIPDATE/7))和已有的列(L_SHIPINSTRUCT)。

输出中接下来的是关于 MQT 的建议，如下所示：

```
-- LIST OF RECOMMENDED MQTs
-- ========================
-- MQT MQT40902204140000 can be created as a refresh immediate MQT
-- mqt[1],   0.009MB
CREATE SUMMARY TABLE "ADVDEMO2"." MQT40902204140000"
    AS(SELECT Q6.CO AS "CO", Q6.C1 AS "C1", …additional details omitted here…)
    DATA INITIALLY DEFERRED REFRESH IMMEDIATE PARTITIONING KEY(C8)
    USING HASHING IN TPCDLDAT ;
COMMIT WORK;
REFRESH TABLE "ADVDEMO2"." MQT40902204140000";
COMMIT WORK;
RUNSTATS ON TABLE "ADVDEMO2"." MQT40902204140000";
COMMIT WORK;
-- MQT MQT409022041530000 can be created as a refresh immediate MQT
(… DDL to create this table follows…)
```

MQT 建议包括：估计的大小、使用的表空间、分区键(如果适用的话)、刷新类型(立即或延迟)以及这个表是否是基本表的复制品(由 REPLICATE 关键字表明)，但在本案例中不是。

最后，Design Advisor 以下面显示的信息结束：

```
8604 solutions were evaluated by the advisor
DB2 Workload Performance Advisor tool is finished.
```

对于经常使用的查询以及大型或复杂的查询，将这些 SQL 语句作为输入，使用 Design Advisor 来建议索引。

此外，针对由完整应用程序测试填充的动态 SQL 缓存重新运行 Design Advisor。这允许根据实际工作负载和 SQL 语句的执行频率建议索引。确保在 Design Advisor 执行之前运行了 RUNSTATS。

12.3　基准测试工具 db2batch

12.3.1　db2batch

基准测试是从各种不同方面(例如数据库响应时间、CPU 和内存使用情况)对应用程序进行评测的过程。基准测试基于可重复的环境，以便能够在相同的条件下运行相同的测试。之后，对测试收集到的结果可以进行评估和比较。

db2batch 是一种基准测试工具，它以一组 SQL 和/或 XQuery 语句作为输入，动态地准备语句和描述语句，并返回结果集。取决于 db2batch 命令中使用的选项，结果集可以返回这些语句的执行时间、关于内存使用情况(例如缓冲池)的数据库管理器快照和缓存信息。

可以在 flat 文件或标准输入中指定要运行基准测试的语句。在输入文件中可以设置很多控制选项。指定这些选项的语法是：--#SET *control_option value*。下面是包含控制选项的一个输入文件的例子：

```
-- db2batch.sql
-- ------------
--#SET PERF_DETAIL 3
--#SET ROWS_OUT 5
-- This query lists employees, the name of their department
-- and the number of activities to which they are assigned for
-- employees who are assigned to more than one activity less than
-- full-time.
--#COMMENT Query 1
```

```
select lastname, firstnme,  deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
     employee.empno = emp_act.empno and    emp_act.emptime < 1
        group by lastname, firstnme, deptname having count(*) > 2;
--#SET PERF_DETAIL 1
--#SET ROWS_OUT 5
--#COMMENT Query 2
select lastname, firstnme,  deptname, count(*) as num act
from employee, department, emp act
where employee.workdept = department.deptno and
     employee.empno = emp act.empno and     emp act.emptime < 1
group by lastname, firstnme, deptname having count(*) <= 2;
```

- 选项 PERF_DETAIL 3 意味着将返回关于花费的时间和数据库管理器、数据库及应用程序的快照这些性能方面的细节。
- 选项 ROWS_OUT 5 意味着无论查询返回的实际行数是多少，只从结果集中取 5 行。
- COMMENT Query1 将语句命名为 Query1。

下面的命令在 sample 数据库上调用基准测试工具，输入文件为 db2batch.sql。

```
db2batch -d sample -f db2batch.sql
```

这个命令将返回查询的结果集(限 5 行)和查询花费的时间及 CPU 时间。另外还返回数据库管理器、数据库和应用程序快照。由于输出很大，因此这里只显示 db2batch 命令的概要。

```
* Summary Table:
Type      Number     Repetitions Total Time(s) Min Time(s)   ...
--------- ----------- ----------- -------------- --------------
Statement           1           1       0.052655      0.052655 ...
Statement           2           1       0.004518      0.004518 ...
...Max Time(s)  Arithmetic Mean Geometric Mean Row(s) Fetched Row(s) Output
-------------- -------------- -------------- -------------- -------------
...      0.052655       0.052655       0.052655             5            5
...      0.004518       0.004518       0.004518             8            5

* Total Entries:              2
* Total Time:               0.057173 seconds
* Minimum Time:             0.004518 seconds
* Maximum Time:             0.052655 seconds
* Arithmetic Mean Time:       0.028587 seconds
* Geometric Mean Time:        0.015424 seconds
```

db2batch 命令支持很多选项。这里只列出其中一些选项，让您对这个工具的功能有所了解。

- -m *parameter_file* 用参数值指定用于绑定到 SQL 语句参数占位符的输入文件。
- -r *result_file* 指定存放命令结果的输出文件。
- -i *short*|*long*|*complete* 指定从哪个方面测量花费的时间。*short* 测量运行每条语句所花费的时间。*long* 测量运行每条语句所花费的时间，包括语句之间的开销。*complete* 测量运行每条语句所花费的时间，分别报告准备、执行和取数据的时间。
- -iso 指定语句使用的隔离级别。默认情况下，db2batch 使用 Repeatable Read 隔离级别。

12.3.2 db2batch 基准程序测试分析示例

基准程序的输出应该包括每个测试的标识、程序执行的迭代、语句号和执行的计时。在经过一系列测量之后，基准程序测试结果的摘要可能类似于如下所示：

```
Test     Iter.   Stmt    Timing        SQLStatement
Numbr    Numbr   Numbr   (hh:mm:ss.ss)
     002      05      01      00:00:01.34    CONNECT TO SAMPLE
     002      05      10      00:02:08.15    OPEN cursor 01
     002      05      15      00:00:00.24    FETCH cursor 01
     002      05      15      00:00:00.23    FETCH cursor 01
     002      05      15      00:00:00.28    FETCH cursor 01
     002      05      15      00:00:00.21    FETCH cursor 01
     002      05      15      00:00:00.20    FETCH cursor 01
     002      05      15      00:00:00.22    FETCH cursor 01
     002      05      15      00:00:00.22    FETCH cursor 01
     002      05      20      00:00:00.84    CLOSE cursor 01
     002      05      99      00:00:00.03    CONNECT RESET
```

注意：
上面报告中的数据仅供说明之用，不表示测量的结果。

分析显示：CONNECT(语句 01)用去 1.34 秒，OPEN cursor(语句 10)用去 2 分钟 8.15 秒，FETCHES(语句 15)返回 7 行且延迟时间最长的为 0.28 秒，CLOSE cursor(语句 20)用去 0.84 秒，而 CONNECT RESET(语句 99)用去 0.03 秒。

如果您的程序可以定界 ASCII 格式输出数据，那么可以在以后将该数据导入数据库表或电子表格以进行进一步的统计分析。

基准程序报告的样本输出可能是：

```
PARAMETER          VALUES  FOR  EACH  BENCHMARK  TEST
TEST  NUMBER        001      002      003     004      005
locklist           63       63       63      63       63
maxappls           8        8        8       8        8
applheapsz          48       48       48      48       48
dbheap             128      128      128     128      128
sortheap           256      256      256     256      256
maxlocks           22       22       22      22       22
stmtheap           1024     1024     1024    1024     1024
SQLSTMT         AVERAGE  TIMINGS(seconds)
    01              01.34    01.34    01.35    01.35    01.36
    10              02.15    02.00    01.55    01.24    01.00
    15              00.22    00.22    00.22    00.22    00.22
    20              00.84    00.84    00.84    00.84    00.84
    99              00.03    00.03    00.03    00.03    00.03
```

注意：

上面报告中的数据仅供说明之用，不表示任何测量的结果。

12.4 数据一致性检查工具

12.4.1 db2dart 及案例

可以使用 db2dart 命令来验证数据库及其对象的体系结构是否正确。还可以使用它来显示数据库控制文件的内容，以便从其他情况下可能无法访问的表中抽取数据。

要显示所有可能的选项，只需发出不带任何参数的 db2dart 命令。如果命令行中未显式指定一些需要参数的选项(如表空间标识)，那么会提示输入这些参数。

默认情况下，db2dart 实用程序将创建名为 databaseName.RPT 的报告文件。对于单一分区数据库分区环境，将在当前目录中创建该文件；对于多分区数据库分区环境，将在诊断目录的子目录中创建该文件。该子目录称为 DART####，其中####是数据库分区号。

db2dart 实用程序通过直接从磁盘中读取数据库中的数据和元数据来对其进行访问。因此，决不能对仍具有活动连接的数据库运行该工具。如果存在活动连接，那么该工具将不知道缓冲池中的页面或内存中的控制结构(此处是举例说明)，可能会报告假的错误结果。同样，如果对需要进行崩溃恢复或尚未完成前滚恢复的数据库运行 db2dart，那么由于磁盘上的数据性质不一致，可能会导致类似的不一致情况。

DBA 可以用 db2dart 对数据库做一致性检查，从而检查数据页和索引页是否有损坏，db2dart 是数据库级别的检查和修复工具。可以检查表空间和表的完整性，以及检查数据页

的页连续一致性，类似 Sybase 数据库的 dbcc 工具。使用 db2dart 时，必须断开所有与数据库的连接，下面举一个例子：

```
    C:\>db2dart SAMPLE /db
    FYI: An active connection to the database has been detected.
         False errors may be reported.
         Deactivate all connections and re-run to verify.
    Warning: The database state is not consistent.
    Warning: Errors reported about reorg rows may be due to the inconsistent
state of the database.
                       DB2DART Processing completed with warning(s)!
                            Complete DB2DART report found in:
    C:\DOCUME~1\ALLUSE~1\APPLIC~1\IBM\DB2\DB2COPY1\DB2\DART0000\SAMPLE.RPT
```

我们可以查看 db2dart 生成的报告文件，对数据库的一致性进行检查，判断数据库中是否有坏的数据页或索引页。关于 db2dart 工具还有很多高级内容。在此书中限于篇幅，我对 db2dart 和 inspect 工具不做过多的讲述。

12.4.2　inspect 及案例

inspect 是 db2dart 的派生命令。inspect 集成在 DB2 引擎中，因此可以利用 DB2 的 bufferpool、数据预取等优势来获得命令和更好的执行性能。inspect 在操作的过程中访问数据库对象，在检查数据库对象过程中使用的隔离级别是 Uncommitted Read。

inspect 会将检查出来的信息放在 dbm 的诊断数据的目录里面，如果在检查工作的最后没有发现错误，那么 inspect 就会自己将文件删除。如果发生错误，那么此文件将被保留。当然，DB2 的这些实用工具的输出都是未格式化的信息，需要进行格式化才能被阅读和分析，格式化的命令是 db2inspf。下面举个例子，在 Windows 平台上使用该命令检查表 emp_resume 的存储情况：

```
    1) db2 inspect check table name emp_resume results keep res.ins
    (注：res.ins 文件将产生在 DB2 诊断日志所在路径下，如 C:\Documents and Settings\All
Users\Application Data\IBM\DB2\DB2COPY1\DB2 目录下)。
    2) db2inspf  res.ins res.txt  ----(格式化 INSPECT 的输出文件)，查看 resume.txt
文件，发现其中有类似以下输出：
    DATABASE: SAMPLE
    VERSION : SQL09075
    2012-09-08-16.57.32.133889
    Action: CHECK TABLE
    Schema name: DB2ARA
    Table name: EMP RESUME
    Tablespace ID: 2  Object ID: 8
```

```
    Result file name: res.ins
         Table phase start (ID Signed: 8, Unsigned: 8; Tablespace ID: 2) :
DB2ARA.EMP_RESUME
           Data phase start. Object: 8  Tablespace: 2
          The index type is 2 for this table.
           Traversing DAT extent map, anchor 672.
           Extent map traversal complete.
           DAT Object Summary: Total Pages 1 - Used Pages 1 - Free Space 76 %
          Data phase end.
            Index phase start. Object: 8  Tablespace: 2
           Traversing INX extent map, anchor 864.
           Extent map traversal complete.
           INX Object Summary: Total Pages 3 - Used Pages 3
          Index phase end.
           LOB phase start. Object: 8  Tablespace: 2
           Traversing LOB extent map, anchor 736.
           Extent map traversal complete.
           Traversing LBA extent map, anchor 800.
           Extent map traversal complete.
           LOB Object Summary: Total Pages 32 - Used Pages 3
           LBA Object Summary: Total Pages 2 - Used Pages 2
          LOB phase end.
        Table phase end.
    Processing has completed. 2012-09-08-16.57.32.198031
```

12.5 db2look

12.5.1 db2look 概述

db2look 是可以从命令行提示符和控制中心调用的强大工具。这个工具可以：

- 从数据库对象中提取数据库定义语言(DDL)语句。
- 生成 UPDATE 语句，用于更新数据库管理器和数据库配置参数。
- 生成 db2set 命令，用于设置 DB2 概要注册表。
- 提取和生成数据库统计报告。
- 生成 UPDATE 语句，用于复制关于数据库对象的统计信息。

LOAD 之类的实用程序要求目标表已经存在。可以使用 db2look 提取表的 DDL，在目标数据库上运行，然后调用装载操作。db2look 非常容易使用，下面的例子展示了这一点。这个命令生成 *peter* 在数据库 department 中创建的所有对象的 DDL，输出被存储在 alltables.sql 中。

```
db2look -d department -u peter -e -o alltables.sql
```

下面的命令生成：

- 数据库 department 中所有对象的 DDL(由-d、-a 和-e 选项指定)。
- UPDATE 语句，用于复制数据库中所有表和索引的统计信息(由选项-m 指定)。
- GRANT 授权语句(由选项-x 指定)。
- 用于数据库管理器和数据库配置参数的 UPDATE 语句和用于概要注册表的 db2set 命令(由选项-f 指定)。

```
db2look -d department -a -e -m -x -f -o db2look.sql
```

db2look 还可以生成用于注册 XML 模式的命令。下面的例子生成模式名为 db2inst1 的对象所需的 REGISTER XMLSCHEMA 和 COMPLETE XMLSCHEMA 命令(由选项-xs 指定)。/home/db2inst1 中将创建输出 db2look.sql，这个目录由-xdir 选项指定。

```
db2look -d department -z db2inst1 -xs -xdir /home/db2inst1 -o db2look.sql
```

生成缓冲池、表空间和数据库分区组信息

```
db2look -d <dbname> -l -o storage.out
```

下面是对以上 db2look 命令中所用选项的描述：

- -d：数据库名——该选项必须指定。
- -l：生成数据库布局。这是用于数据库分区组、缓冲池和表空间的布局。
- -o：将输出重新定向到给定的文件名。如果未指定 -o 选项，然么输出将为标准输出(stdout)，通常是输出到屏幕。

创建数据定义语言(DDL)

下列 db2look 命令创建 DDL 以复制所有数据库对象，以及配置和统计信息：

```
db2look -d <dbname> -e -a -m -o db2look.out
```

这里，我们使用了下列参数：

- -a：为所有的创建器(creator)生成统计数据。如果指定了该选项，那么将忽略-u 选项。
- -e：提取复制数据库所需的 DDL 文件。该选项生成包含了 DDL 语句的脚本。该脚本可以在另一数据库上运行以重新创建数据库对象。

- -m：以模拟模式运行 db2look 实用程序。该选项生成包含了 SQLUPDATE 语句的脚本。这些 SQLUPDATE 语句捕获所有的统计数据。该脚本可以在另一数据库上运行以复制原来的那个数据库。当指定-m 选项时，将忽略-p、-g 和-s 选项。

收集数据库子集的统计数据和 DDL

为了仅仅收集某些表和相关对象的统计数据和 DDL，可使用下列命令：

```
db2look -d <dbname> -e -a -m -t <table1> <table2> .. <tableX> -o table.ddl
```

这里，我们使用了下列附加参数：

- -t：为特定的表生成统计数据。可以将表的最大数目指定为 30。

此外，如果不使用-a 选项，就可以使用-z 选项：

- -z：模式名。如果同时指定了-z 和-a，那么将忽略-z。

12.5.2 利用 db2look 构建模拟测试数据库

要想构建模拟生产环境的测试数据库，我们需要用到 db2look 下面几个选项：-l 选项导出数据库布局；-m 选项导出数据库统计信息；-f 选项导出数据库配置文件；-fd 选项在测试环境内存资源不足的情况下模拟和生产环境同样的内存。

-l 选项

-l 选项对于模拟生产环境十分重要。理想情况下，您需要具有相同的缓冲池、数据库分区组(如果处于多分区环境中)和表空间信息(包括临时表空间)。但是，如果受到内存约束，无法分配生产中所需具有的大型缓冲池，那么就使用 db2fopt 命令。

当然并非总是可以在测试中设置与生产中相同的表空间。例如，可能设置了大型设备，却无法灵活地在测试中创建相同的设备大小。或者，可能根本无法在测试环境中获得单独的表空间设备。此外，或许无法在测试中设置与生产中相同的路径。需要适当地更改路径、设备和文件以适应测试环境。

下面是优化器为表空间所使用的重要信息。这就是您需要确保在测试和生产中相同的信息(**注意**：这里展示的数字是一个例子，您应在测试中使用与生产中相同的设置)。

```
PREFETCHSIZE 16
EXTENTSIZE 16
OVERHEAD 12.670000
TRANSFERRATE 0.180000
```

如果生产中表空间是“由数据库管理的”，那么在测试中也应该是“由数据库管理的”。如果在生产中是“由系统管理的”，那在测试中也应该是这样的方式。

-m 选项

-m 选项极其重要。该选项从系统表收集所有统计数据。测试中的统计数据必须与生产中的相同，这些统计数据是可以在测试环境中模拟生产环境的关键。

-f 和-fd 选项

```
db2look -d <dbname> -f -fd -o config.out
```

- -f：提取配置参数和注册变量。如果指定了该选项，就会忽略-wrapper 和-server 选项。
- -fd：为 opt_buffpage 和 opt_sortheap 生成 db2fopt 语句，以及其他配置和注册表设置。

该命令的输出如下所示：

```
$db2look -d sample -f -fd
-- No userid was specified, db2look tries to use Environment variable USER
-- USER is: DB2ARA
-- This CLP file was created using DB2LOOK Version "9.7"
-- Timestamp: Sat Sep  8 18:15:49 GMT+08:00 2012
-- Database Name: SAMPLE
-- Database Manager Version: DB2/AIX64 Version 9.7.5
-- Database Codepage: 1208
-- Database Collating Sequence is: IDENTITY
-- Binding package automatically ...
-- Bind is successful
-- Binding package automatically ...
-- Bind is successful

CONNECT TO SAMPLE;
--------------------------------------------------------
-- Database and Database Manager configuration parameters
--------------------------------------------------------
UPDATE DBM CFG USING cpuspeed 2.991513e-07;
UPDATE DBM CFG USING intra parallel NO;
UPDATE DBM CFG USING comm bandwidth 100.000000;
UPDATE DBM CFG USING federated YES;
UPDATE DBM CFG USING fed_noauth NO;

!db2fopt SAMPLE update opt_buffpage 50000;
!db2fopt SAMPLE update opt_sortheap 10000;
UPDATE DB CFG FOR SAMPLE USING locklist 4096;
UPDATE DB CFG FOR SAMPLE USING dft_degree 1;
UPDATE DB CFG FOR SAMPLE USING maxlocks 10;
```

```
UPDATE DB CFG FOR SAMPLE USING avg_appls 1;
UPDATE DB CFG FOR SAMPLE USING stmtheap 8192;
UPDATE DB CFG FOR SAMPLE USING dft_queryopt 5;
UPDATE DB CFG FOR SAMPLE USING cur_commit ON;
---------------------------------
-- Environment Variables settings
---------------------------------
COMMIT WORK;
CONNECT RESET;
TERMINATE;
```

-f 和-fd 选项是用于提取配置参数和注册变量的关键选项，而优化器将在访问计划阶段使用这些配置参数和环境。在上面的示例中，请注意-fd 选项产生的下列输出：

```
!db2fopt SAMPLE update opt buffpage 50000;
!db2fopt SAMPLE update opt_sortheap 10000;
```

db2fopt 命令告诉优化器为“缓冲池大小”(buffer pool size)使用指定的值，而非将可用缓冲池变量的页面加起来。例如，假设由于测试系统上的内存约束，而导致您无法获得大型缓冲池。您希望将大小配置得相同，实际上却并非有这么大。使用将生成必要的 db2fopt 命令的-fd 选项来告诉优化器使用指定大小，而非基于对数据库可用的缓冲池进行计算。

12.6 其他工具

12.6.1 db2bfd

db2bfd(DB2 Bind File Description)可以查看静态嵌入 SQL 的绑定文件的头部、宿主变量和时间戳时间，它对我们定位－818 错误非常有帮助。请看下面的例子：

```
DS8K:/db2/db2ara/sqllib/bnd$db2bfd -b -s -v  tbscont.bnd
----  -b 显示绑定文件头信息   -s 显示 SQL 语句信息  -v 显示变量声明信息
tbscont.bnd:  Header Contents
Header Fields:
Field   Value
-----   -----
releaseNum      0x800
Endian  0x42
numHvars        9
maxSect         4
numStmt         14
optInternalCnt  4
```

```
    optCount        10
    Name                 Value
    ------------------  -----
    Isolation Level     Uncommitted Read
    Creator              "NULLID  "
    App Name             "TBSCONT " ------程序包名称
    Timestamp            "111017:User defined timestamp"  ----绑定时间戳
    Cnulreqd             Yes
    Sql Error            No package
    Block                Block All
    Validate             Bind
    Date                 Default/local
    Time                 Default/local
    *** All other options are using default settings as specified by the server
***
    tbscont.bnd:  SQL Statements = 14  ----------------SQL 语句
    Line Sec Typ Var Len SQL statement text
    ---- --- --- --- ---
------------------------------------------------------
     117   0   5   0  21 BEGIN DECLARE SECTION
     127   0   2   0  19 END DECLARE SECTION
      ..................很多 SQL 语句，此处略......................
     311   4   0   1 107 SELECT colcount INTO :H00004 FROM SYSIBM.SYSTABLES
                          WHERE creator = 'SYSIBM' AND name='SYSTABLES'
    tbscont.bnd: Host Variables = 9
    Type SQL Data Type     Length Alias  Name Len Name           UDT Name
    ---- --------------    ------ ------ -------- ---------------
-------------
     460  C  STRING            129 H00001       8 database
    ......
     460  C  STRING             19 H00009       8 password
```

12.6.2　db2_kill 和 db2nkill

在 UNIX 和 Linux 上，如果无法正常停止实例，可以使用 db2_kill 杀掉后台进程，注意做完后最好用 ipcrm 或 ipclean 清除共享内存资源。在 Windows 上对应的命令是 db2nkill。

12.6.3　db2tbst

当表空间出现异常状态时，可以使用 db2tbst 查看详细的状态信息。看下面的例子：

```
    C:\Program Files\IBM\SQLLIB\samples\c>db2tbst -0x0c000000
    State = StorDef is in Final State
```

```
         + DMS Rebalance in Progress
         + Tablespace Deletion in Progress
         + Tablespace Creation in Progress
```

12.7 本章小结

“君欲善其事，必先利其器”，本章我们给大家讲解了 DB2 中很多功能强大的工具。这些工具能够在某个方面帮我们发现、诊断、分析并辅助解决问题。



DB2 V10.1 新特性

2012 年春，IBM 成功发布了全新的 DB2 版本——V10.1，相比较市面上广为使用的 DB2 V9.7 和 DB2 V9.5，这次全新的 DB2 版本对旧版本数据库而言可谓彻底的颠覆，不但增加了很多全新的功能，对原有功能也进行了改进。最为重要的是，pureScale 技术更加成熟和完善，已经从 DB2 V9.8 里的实验性技术转变为客户可以正式使用的稳定产品。

熟话说“技多不压身”，那么经历多年的锤炼，修炼了多门顶级功夫的 DB2 V10.1 系列产品，能否颠覆 Oracle 在开源领域的霸主地位，让我们拭目以待。本章主要讲解以下功夫：

- 分身大法——pureScale
- 九阴白骨爪——Continue Data Ingest
- 缩骨大法——自适应压缩
- 乾坤大挪移——灾备功能增强
- 凌波微步——性能增强
- 火眼金睛——监视功能增强
- 金钟罩——安全功能增强

13.1 分身大法——pureScale

13.1.1 基本介绍

“会分身术者，能以一身分出几身，几十身，乃至千百身”。在数据库领域，分身大法指的是应用可伸缩性，允许通过“双机(active-active)”配置将数据库扩展到一组服务器上，以便交付高水平的可用性和可伸缩性。在这种配置中，运行于各主机(或服务器)上的 DB2 副本可以同时读取和写入相同的数据。

在谈 DB2 pureScale 之前，我们不得不先谈一下 IBM DB2 for z/OS，因为 DB2 pureScale 的思路和架构几乎完全来自于 DB2 for z/OS。当今没有任何一款数据库产品能够在扩展性和高可用性方面与 DB2 for z/OS 相提并论，原因是 DB2 for z/OS 采用的底层技术可以确保服务器持续满足 SLA 的要求，其 Coupling Facility 技术提供了集中的锁处理和全局缓存机制，这为服务器扩展和高速恢复提供了可能。因此，DB2 pureScale 完全继承了 DB2 for z/OS 的优良传统，使用软件实现了类 z/OS 的功能，同时这些软件借助于直接内存读写(RDMA)技术，绕过 CPU 中断处理，完全可以替代 z/OS 的硬件底层实现，实现大吞吐量和高响应时间。

与其说 DB2 pureScale 是一项 IBM 技术，不如说 DB2 pureScale 是 DB2 的一种架构方式，同时这种架构方式可以归类为 IBM 对 Share Disk 的一种具体实现。其最基本的结构如图 13-1 所示。

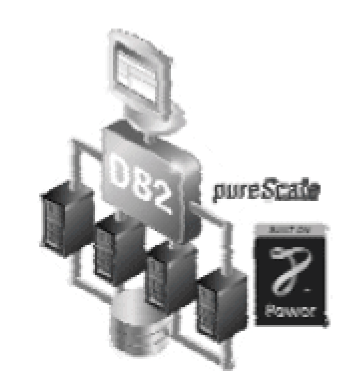


图 13-1　pureScale 结构

在这种结构下，pureScale 具有如下 3 个基本特点：

- 无限的扩展能力：DB2 pureScale 最多支持 128 个节点，同时增加节点对性能提升计划是线性的。图 13-2 是 IBM 在提供的不同 Member 数量时，pureScale 服务器性能增长的线性关系。在没有硬件瓶颈的情况下，性能损失是非常少的。

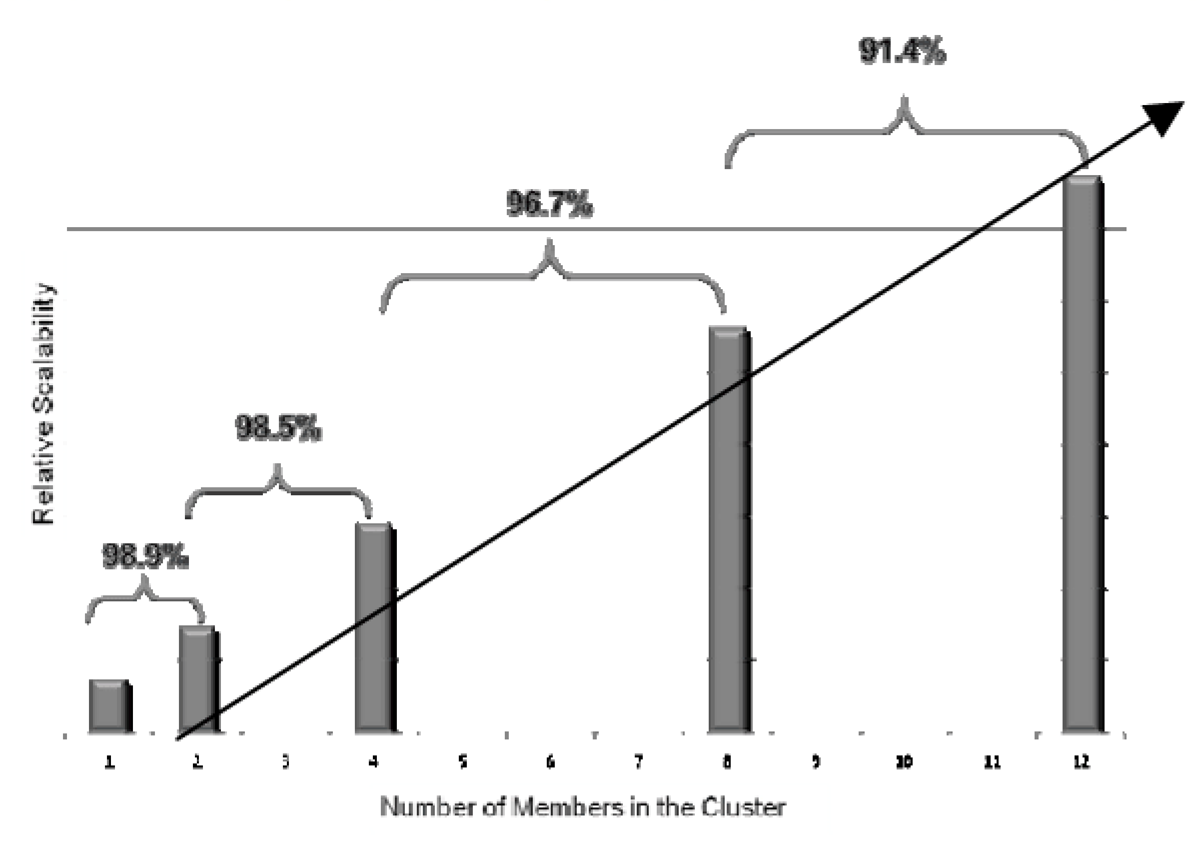


图 13-2　pureScale 性能曲线

- 应用透明：应用无须改动即可支持 pureScale 架构，同时扩展节点对应用无影响。
- 持续的高可用性：单个节点失效对整体的应用几乎无影响，并且可以进行快速恢复。

下面我们再来看一下 purScale 的基本组件构成，图 13-3 显示了 purScale 的基本组件以及各组件之间的关系。其中包含的对象包括：

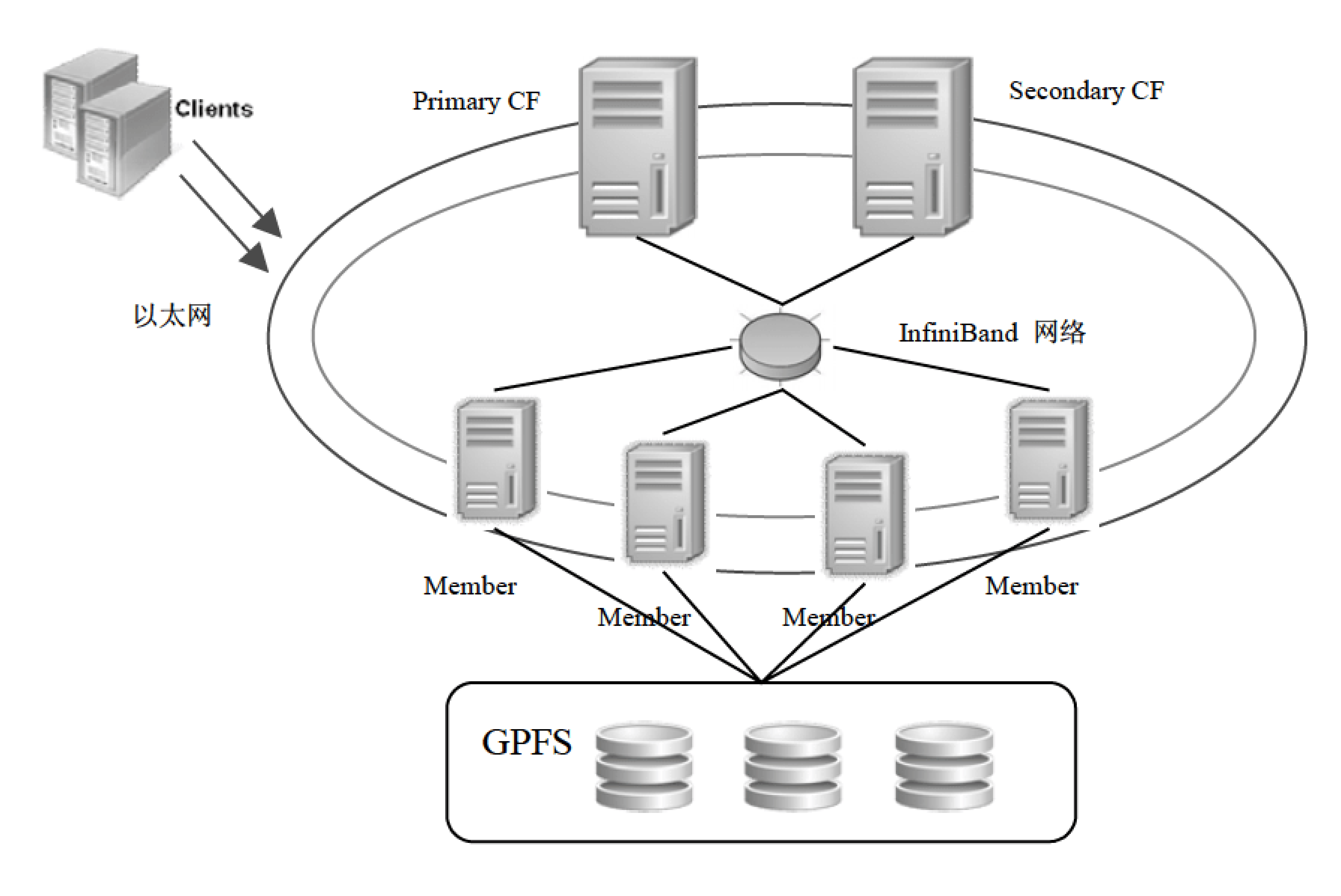


图 13-3　pureScale 架构的基本组件

- Primary CF：CF 的全称是 Cluster Facilitator，用来确保各节点数据的一致性，可以配置一个或两个 CF。CF 的主要组件包括以下 3 部分：
  - ◊ Global Buffer Pool(GBP)：保证集群上缓存中共享数据页的一致性。
  - ◊ Global Lock Manager(GLM)：保证集群上共享数据更改的一致性。
  - ◊ Shared Communications Area(SCA)：对 DB2 控制数据(比如控制块、日志序列数(LSN))提供一致机制。
- Secondary CF：备用 Cluster Facilitator，和 Primary CF 实时同步，防止 Primary CF 单点故障。
- Member：相当于 DPF 的 partition，每个 Member 上有独立的 db2sysc 进程和内存空间，独立写事务日志，共享数据库的数据，可以动态扩展。

- InfiniBand 网络：各 CF 和 Member 之间交互的网络，需要安装专用的 InfiniBand 卡和 InfiniBand 交换机，支持 RDMA 协议。
- 以太网络：DB2 对外连接的网络，每个客户端应用通过以太网络连接到任意 Member。

在图 13-3 中，总共有 2 个 CF 和 4 个 Member，数据库磁盘使用 GPFS 的网络存储技术将数据库文件共享给所有 Member，每个 Member 拥有本地的缓冲池、内存区和日志文件，CF 负责维护全局缓存、全局锁和控制信息。应用可以通过和原 ESE 一样的技术连接到任意 Member 上，数据库会根据负载动态地分发请求给不同的 Member 以实现负载均衡。Member 与 CF 的交互采用一种轻量级的内存直接读写技术进行交互，无需上下文切换，无需 IP 栈处理，甚至无需让出 CPU 时间，保证了节点间交互的低损耗和高性能。

13.1.2 安装和管理

pureScale 有下面几种安装方式，如图 13-4 所示。对于第一种方式，CF 和 Member 可以部署在同一台服务器上，这种方式适用于压力不大的小型 OLTP 系统；对于第二种方式，CF 和 Member 单独部署在不同的机器上，这保证了 CF 的性能，适用于压力比较大的大型 OLTP 系统。

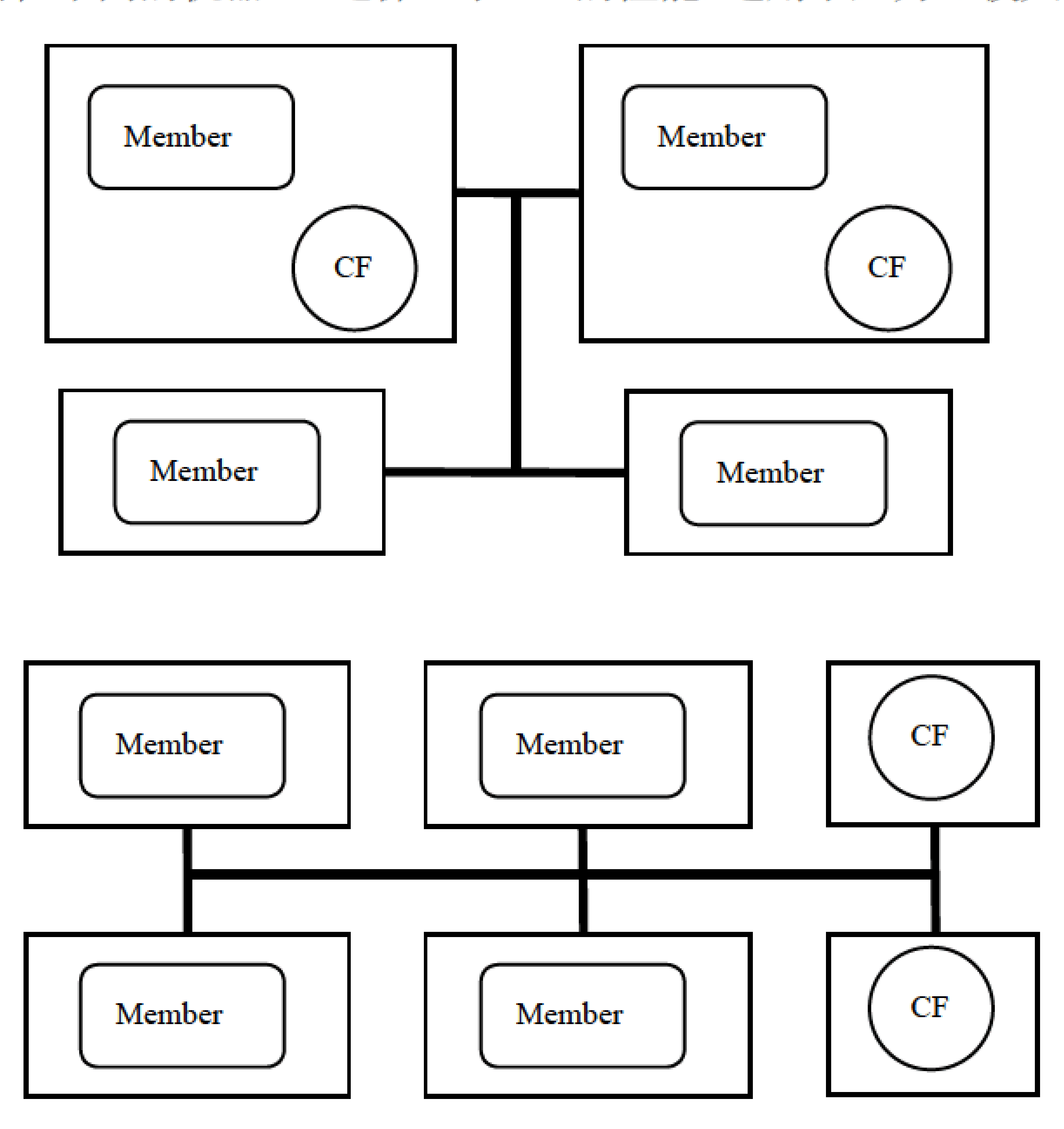


图 13-4　pureScale 的安装方式

pureScale 本身的安装过程非常简单，DB2 已经集成了大部分的安装过程，但是需要提前做好准备工作，包括

- 基本的硬件和网络环境。
- 操作系统满足要求。
- 配置 InfiniBand 驱动和网络。
- 配置 NTP 服务。
- 磁盘规划。

具体要求请查看最新的 DB2 信息中心，这里就不展开了。在这些前提条件准备好后，就可以安装数据库软件。数据库软件只需要随便调一台机器安装即可，通常采用 db2_install 命令行进行安装：

```
db2_install -b <install_path> -p ESE
```

安装完成后，可以用/usr/local/bin/db2ls 检查安装是否成功：

```
    #/usr/local/bin/db2ls
    Install Path                       Level   Fix Pack   Special Install Number
Install Date                  Installer UID
    ---------------------------------------------------------------------------
    /opt/IBM/db2/V10.1                10.1.0.0       0
Sat Sep  8 10:46:15 2012 BEIST            0
```

在初始节点上安装了 DB2 pureScale 之后，接下来可以创建和配置 DB2 pureScale 实例。DB2 pureScale 提供了集成的命令 db2cluster_prepare 来创建 DB2 pureScale 实例共享目录的 GPFS 文件系统：

```
./db2cluster_prepare -instance_shared_dev /dev/<diskname>
```

-instance_shared_dev 的后面指定 GPFS 文件系统里实例目录使用的磁盘号。

该命令执行完之后会创建文件系统，名称自动为 db2fs1，挂载点在/db2sd_<timestamp>，挂载点名称也是自动生成的，命令完成后可以用 df 验证：

```
    #df -g
    /dev/db2fs1      100.00     99.24    1%     4416     4%
/db2sd_20120908150547
```

还可以用 GPFS 自己的命令 mmlsfs 或 mmlsnsd 来验证 GPFS 文件系统和 NSD 盘：

```
#mmlsfs
File system attributes for /dev/db2fs1:
```

```
    =======================================
    flag                value                     description
    ------------------- ------------------------
-----------------------------------
     -f                 32768                     Minimum fragment size in bytes
     -i                 512                       Inode size in bytes
     -I                 32768                     Indirect block size in bytes
     -m                 1                         Default number of metadata replicas
     -M                 2                         Maximum number of metadata replicas
     -r                 1                         Default number of data replicas
     -R                 2                         Maximum number of data replicas
     -j                 cluster                   Block allocation type
     -D                 nfs4                      File locking semantics in effect
     -k                 all                       ACL semantics in effect
     -n                 130                       Estimated number of nodes that will
                                                  mount file system
     -B                 1048576                   Block size
     -Q                 none                      Quotas enforced
                        none                      Default quotas enabled
     --filesetdf          no                       Fileset df enabled?
     -V                 12.10(3.4.0.7)            File system version
     --create-time        Sat Sep  8 15:05:56 2012 File system creation time
     -u                 yes                       Support for large LUNs?
     -z                 no                        Is DMAPI enabled?
     -L                 4194304                   Logfile size
     -E                 yes                       Exact mtime mount option
     -S                 no                        Suppress atime mount option
     -K                 whenpossible               Strict replica allocation option
     --fastea            yes                       Fast external attributes enabled?
     --inode-limit        135168                    Maximum number of inodes
     -P                 system                    Disk storage pools in file system
     -d                 gpfs1nsd                  Disks in file system
     -A                 yes                       Automatic mount option
     -o                 none                      Additional mount options
     -T                 /db2sd_20120908150547    Default mount point
     --mount-priority    0                         Mount priority
    #mmlsnsd
    File system   Disk name    NSD servers
    -----------------------------------------------------------------------
    db2fs1       gpfs1nsd    (directly attached)
```

接下来可以用 db2icrt 来创建 DB2 pureScale 实例：

```
./db2icrt -d -a SERVER ENCRYPT -p <port> -m MemberHostName -mnet MemberNetname
-cf CFHostName -cfnet CFNetname -instance shared dir /db2sd <timestamp> -tbdev
/dev/<hdisk> -s dsf -u <fence user> <instnace user>
```

该命令各个参数的意义如下：

-d：开启 debug 模式，获得更多的诊断信息。

-a：实例的认证方式。

-p：实例将采用的服务端口。

-m：集群的第一个成员节点主机名。

-mnet：InfiniBand 网卡接口的主机名。

-instance_shared_dir：实例的共享文件目录。

-tbdev：集群的 tiebreaker disk。

-s：实例的类型。

-u：fenced 用户名。

db2icrt 执行完之后，DB2 pureScale 集群实例已经被创建出来，只不过现在集群中只有一个成员节点和一个 CF 节点。接下来需要添加剩下的成员节点和 CF 节点。在添加其他的节点时，如果节点所在机器上并没有安装 DB2 的软件，那么 db2iupdt 命令会自动将初始节点的二进制文件拷贝到要添加节点的机器上，因此第一次添加节点的时间相对会比较长。

添加第二个 CF 节点：

```
    ./db2iupdt -add -cf CFHostName -cfnet CFNetname <instance name>
```

依次添加后续的成员节点：

```
    ./db2iupdt -add -m MemberHostName -mnet MemberNetname <instance name>
```

为了支持 TCP 访问，我们还需要切换至实例用户以设置 DB2COMM 环境变量：

```
    db2set DB2COMM=tcpip
```

至此，DB2 pureScale 的集群和实例创建已经完成，可以切换到实例用户，利用 db2instance -list 命令查看集群状态：

```
    $db2instance -list
    ID        TYPE             STATE              HOME HOST
CURRENT HOST           ALERT   PARTITION NUMBER       LOGICAL PORT    NETNAME
    --        ----             -----              ---------
------------           -----   ----------------       ------------    -------
```

```
    1       MEMBER          STARTED                  pure2
pure2               NO                0                0     pure2-ib0
    2       MEMBER          STARTED                  pure3
pure3               NO                0                0     pure3-ib0
    128     CF             PRIMARY                   pure1
pure1               NO                -                0     pure1-ib0
    129     CF               PEER                   pure4
pure4               NO                -                0     pure4-ib0

    HOSTNAME                 STATE              INSTANCE STOPPED       ALERT
    --------                 -----              ----------------       -----
      pure4                 ACTIVE                          NO          NO
      pure1                 ACTIVE                          NO          NO
      pure3                 ACTIVE                          NO          NO
      pure2                 ACTIVE                          NO          NO
```

创建好实例后，就可以继续创建数据库。在创建数据库之前，首先调用 db2cluster -create 命令来创建数据库所在的 GPFS 文件系统，该命令如下：

```
./db2cluster -CFS -CREATE -FILESYSTEM <fsname> -DISK /dev/<diskname>, -MOUNT
<mount point>
```

创建完之后，还可以用 db2cluster -list 命令来查看创建的文件系统：

```
    $db2cluster -cfs -list -filesystem
    FILE SYSTEM NAME                     MOUNT_POINT
    ---------------------------------     -------------------------
    db2data                             /db2data
    db2fs1                              /db2sd_20120908150547
```

后面创建数据库和对象的过程和 ESE 版的 DB2 没有区别，只是需要将数据库路径指定到我们刚才创建的 GPFS 文件系统下。比如：

```
    db2 "create database PUREDB AUTOMATIC STORAGE YES ON /db2data USING CODESET
UTF-8 TERRITORY CN"
```

创建好数据库后，可以在任意 Member 节点上连接数据库，进行 SQL 操作，同时 pureScale 会根据负载情况，将数据库请求分发到不同的 Member 节点上以并行执行，整个过程对应用来说完全透明，应用不需要做任何改动就可以从单节点数据库迁移到多节点数据库上。要实现应用的负载均衡功能，对于 Java 平台，只需要在连接字符串中指定 enableSysplexWLB=true 属性即可。

13.1.3　性能监控

为了更好地支持 pureScale 功能，DB2 在原有基础上又增加了许多监控指标和监控视图，这里再简单跟大家介绍一下。

1. 缓冲池监控

上面说过，pureScale 环境下数据库的数据缓冲池分为 GBP 和 LBP 两种。其中，GBP 由 CF 管理，LBP 则由每个 Member 进行管理。GBP 里只存放所有 LBP 里页的指针，用于最终页的变化和修改过的数据，LBP 里存放 Member 经常用到的数据。因此，监控缓冲池命令率也要从 GBP 和 LBP 两方面进行监控。对于缓冲池监控，我们应该关心的指标有：

pool_data_lbp_pages_found
数据页出现在本地缓冲池中的次数。比如我们需要查询一行数据，这行数据如果在本地缓冲池中，那么此指标数加一。

pool_data_gbp_l_reads
尝试从全局缓冲池读取依赖于全局缓冲池(GBP)的数据页(因为该页在本地缓冲池(LBP)中无效或不存在)的次数。比如我们需要查询一行数据，但是如果数据在本地缓冲池中不存在，但是在 GBP 中存在，那么此指标数加一。

pool_data_gbp_p_reads
尝试将磁盘中依赖于全局缓冲池(GBP)的数据页读取到本地缓冲池中(因为在 GBP 中找不到该页)的次数。比如我们需要查询一行数据，但是如果数据在本地缓冲池和全局缓冲池都不存在，需要从磁盘读取，那么此指标数加一。

pool_data_gbp_invalid_pages
数据页在本地缓冲池中无效并因此改为从全局缓冲池中读取的次数。在 DB2 数据共享环境的外部，此值为空。比如我们需要查询一行数据，但是如果数据在本地缓冲池中被标记为无效，需要从全局缓冲池读取，那么此指标数加一。

pool_async_data_gbp_l_reads
预取程序尝试将磁盘中依赖于全局缓冲池(GBP)的数据页读取到本地缓冲池中(因为在 GBP 中找不到该页)的次数。比如我们需要查询一行数据，但是如果数据在本地缓冲池中不存在，DB2 为了优化后续的查询，可能需要从全局缓冲池中一次读取多行数据到本地缓冲池，那么此指标数加一。

表 13-1 可以很好地表示不同情况下指标的增减情况。

表 13-1　缓冲池指标的增减变化

指　　标	在 LBP 中存在	在 LBP 中无效在 GBP 中存在	不在 LBP 中在 GBP 中存在	不在 LBP 或 GBP 中在磁盘上
pool_data_l_reads	+1	+1	+1	+1
pool_data_lbp_pages_found	+1	+1		
pool_data_gbp_l_reads		+1	+1	+1
pool_data_gbp_invalid_pages		+1		
pool_data_gbp_p_reads				+1
pool_data_p_reads				+1

只了解这些指标，是无法帮助我们诊断数据库问题的，我们需要结合一些比率来帮助进行判断，常用的比率如下：

总体命中率

可以使用 pool_data_l_reads 和 pool_data_p_reads 监视元素来计算总体的数据页命中率。例如，以下公式返回 DB2 环境中数据页的总体命中率：

```
(pool_data_l_reads -(pool_data_p_reads - pool_async_data_reads))/
pool_data_l_reads * 100
```

示例如下：

```
$db2 "select substr(bp_name,1,20)as bp_name,
>((pool_data_l_reads-(pool_data_p_reads-pool_async_data_reads))*100/pool_da
ta_l_reads)
> FROM TABLE(MON_GET_BUFFERPOOL('',-2))"

BP_NAME              2
-------------------- --------------------
IBMDEFAULTBP                          83
BP_INX_4K                             85
```

LBP 命中率

可以结合使用 pool_data_lbp_pages_found 和 pool_data_l_reads 来计算本地缓冲池的命中率。例如，以下公式返回 DB2 环境中的本地缓冲池命中率：

```
(pool_data_lbp_pages_found - pool_async_data_lbp_pages_found)/
pool_data_l_reads
```

示例如下：

```
$db2 "select substr(bp name,1,20)as bp name,
>(pool data lbp pages found - pool async data lbp pages found)*100 /
pool data l reads
> FROM TABLE(MON GET BUFFERPOOL('',-2))"

BP NAME              2
-------------------- --------------------
IBMDEFAULTBP                           78
BP_INX_4K                              39
```

GBP 命中率

可以结合使用 pool_data_gbp_l_reads 和 pool_data_gbp_l_reads 来计算全局缓冲池的命中率。例如，以下公式返回 DB2 环境中的全局缓冲池命中率：

```
(pool_data_gbp_l_reads - pool_data_gbp_p_reads)/ pool_data_gbp_l_reads
```

示例如下：

```
$db2 "select substr(bp name,1,20)as bp name,
>(pool data gbp l reads-pool data gbp p reads)*100/pool data gbp l reads
> FROM TABLE(MON GET BUFFERPOOL('',-2))"

BP NAME              2
-------------------- --------------------
IBMDEFAULTBP                            7
BP_INX_4K                              18
```

2. 页回收监控

页回收是 pureScale 里不同成员之间相互交换页面的一种技术，当不同节点需要修改同一页面里的数据(即使位于不同行中)时，就会触发页回收。频繁的页回收行为有可能造成一些内部锁的竞争，影响系统性能，因此页回收监控也是 pureScale 里非常重要的监控指标。DB2 里新增加了 mon_get_page_access_info 函数来监控页回收次数。比如可以通过下面的 SQL 来监控某个 Schema 下每张表的页回收次数：

```
SELECT SUBSTR(TABNAME,1,8)AS NAME,
      SUBSTR(OBJTYPE,1,5)AS TYPE,
      PAGE_RECLAIMS_X AS PGRCX,
      PAGE_RECLAIMS_S AS PGRCS,
      SPACEMAPPAGE_PAGE_RECLAIMS_X AS SMPPGRCX,
      SPACEMAPPAGE_PAGE_RECLAIMS_S AS SMPPGRCS
```

```
FROM TABLE( MON_GET_PAGE_ACCESS_INFO('CHGMDB', NULL, NULL))AS WAITMETRICS
ORDER BY NAME
```

如果发现特别频繁的页回收行为并且严重影响了系统性能，可以通过下面几个通用的方法来解决：

- 减小 PAGESIZE。让每个 PAGE 包含较少的行，这样就减小了不同程序更新同一 PAGE 的可能性。
- 对于更新比较频繁的表，可以采用增大 PCTFREE 的方式，这种方式同样可以让每个 PAGE 包含较少的行。
- 使用 current member 内置函数，使每个应用程序只修改 current member 相同的行。

```
$db2 "alter table test add column curmem smallint default current member
implicitly hidden"
```

3. CF CPU 监控

使用 vmstat 或其他 CPU 检测工具，可以发现 CF 的 CPU 利用率一直在 100%左右，即使在当前集群环境空闲的时候。主要原因是 CF 需要启动一下后台线程实时监控和同步不同节点的状态。因此，DB2 提供了 env_cf_sys_resources 管理视图来更准确地监控 CF 的 CPU 利用率：

```
pure1:/#vmstat 1 10

System configuration: lcpu=8 mem=8192MB

kthr    memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   free  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 7  0 613326 1328839   0   0   0   0    0   0   8  788 1162  94  0  6  0
 7  0 613326 1328839   0   0   0   0    0   0  10 3297 1182  94  0  5  0
 7  0 613326 1328839   0   0   0   0    0   0  12  584 1143  94  0  6  0
 7  0 613326 1328839   0   0   0   0    0   0   9  574 1162  94  0  6  0
 7  0 613326 1328839   0   0   0   0    0   0  45  603 1171  94  0  6  0
 7  0 613326 1328839   0   0   0   0    0   0   9  957 1157  94  0  6  0
 0  0 613326 1328839   0   0   0   0    0   0  10  595 1067  94  0  6  0
```

比如下面这个 SQL 会返回 Primary CF 和 Secondary CF 的 CPU 利用率以及内存利用率：

```
$db2 "SELECT VARCHAR(NAME,20)AS ATTRIBUTE,
> VARCHAR(VALUE,25)AS VALUE, VARCHAR(UNIT,8)AS UNIT FROM
SYSIBMADM.ENV_CF_SYS_RESOURCES"
```

```
ATTRIBUTE            VALUE                     UNIT
-------------------- ------------------------- --------
HOST_NAME            pure1                     -
MEMORY_TOTAL          8192                      MB
MEMORY_FREE          5191                      MB
MEMORY_SWAP_TOTAL     16384                     MB
MEMORY_SWAP_FREE     16370                     MB
VIRTUAL MEM TOTAL     24576                     MB
VIRTUAL MEM FREE     21561                     MB
CPU USAGE TOTAL      0                         PERCENT
HOST NAME            pure4                     -
MEMORY TOTAL          8192                      MB
MEMORY FREE          5437                      MB
MEMORY SWAP TOTAL     16384                     MB
MEMORY SWAP FREE     16370                     MB
VIRTUAL MEM TOTAL     24576                     MB
VIRTUAL MEM FREE     21808                     MB
CPU USAGE TOTAL      0                         PERCENT

  16 record(s)selected.
```

13.2　九阴白骨爪——Continue Data Ingest

话说当年桃花岛铁尸梅超风和铜尸陈玄风夫妇根据《九阴真经》下半部练就的阴毒武技，以十指摧骨破骨，狠辣无比，横扫江湖一时。而今在 DB2 V10.1 中同样有了这样的武器——Continue Data Ingest，可以帮助我们快速导入数据，做到快、准、狠。

13.2.1　Continue Data Ingest 介绍

INGEST 是 DB2 V10.1 的新特性，是为了通过大量持续的数据流来实时和并发地快速进行数据导入而引入的新概念。这样可以避免数据被锁，可以 24 小时对数据库进行操作。INGEST 有数据修复的功能，在数据导入中断时能找到中断点并能继续进行数据导入。INGEST 支持复杂的 SQL 表达式语法，是更加好用的客户端软件。用户可以用 INGEST 做 Insert、Update、Delete、Replace 和 Merge 操作，更方便地为客户提供服务。

INGEST 支持文件和管道两种输入类型，支持 DEL、ASC 两种数据格式，支持 SQL 表达式，支持 Insert、Update、Replace、Delete 和 Merge 操作，支持并发更改、插入，删除，可以根据时间间隔或数据条数来提交数据。INGEST 会把未插入的数据放入文件、表

中或者直接丢弃，支持修复和重启。

INGEST 支持 ESE、PureScale 和 DPF 环境。INGEST 与数据引擎使用标准化的外部接口，是个多线程工具，速度快，效率高。

INGEST 支持很多种表，但不支持 AQT(accelerated query tables)、CGTT/DGTT(created or declared global temporary tables)、typed tables 和 typed views。同时，INGEST 不支持 LOB(BLOB、CLOB 和 DBCLOB)、XML、structured types 以及基于这些数据类型的自定义数据类型。

单个 INGEST 命令将经历 3 个主要阶段，如图 13-5 所示。

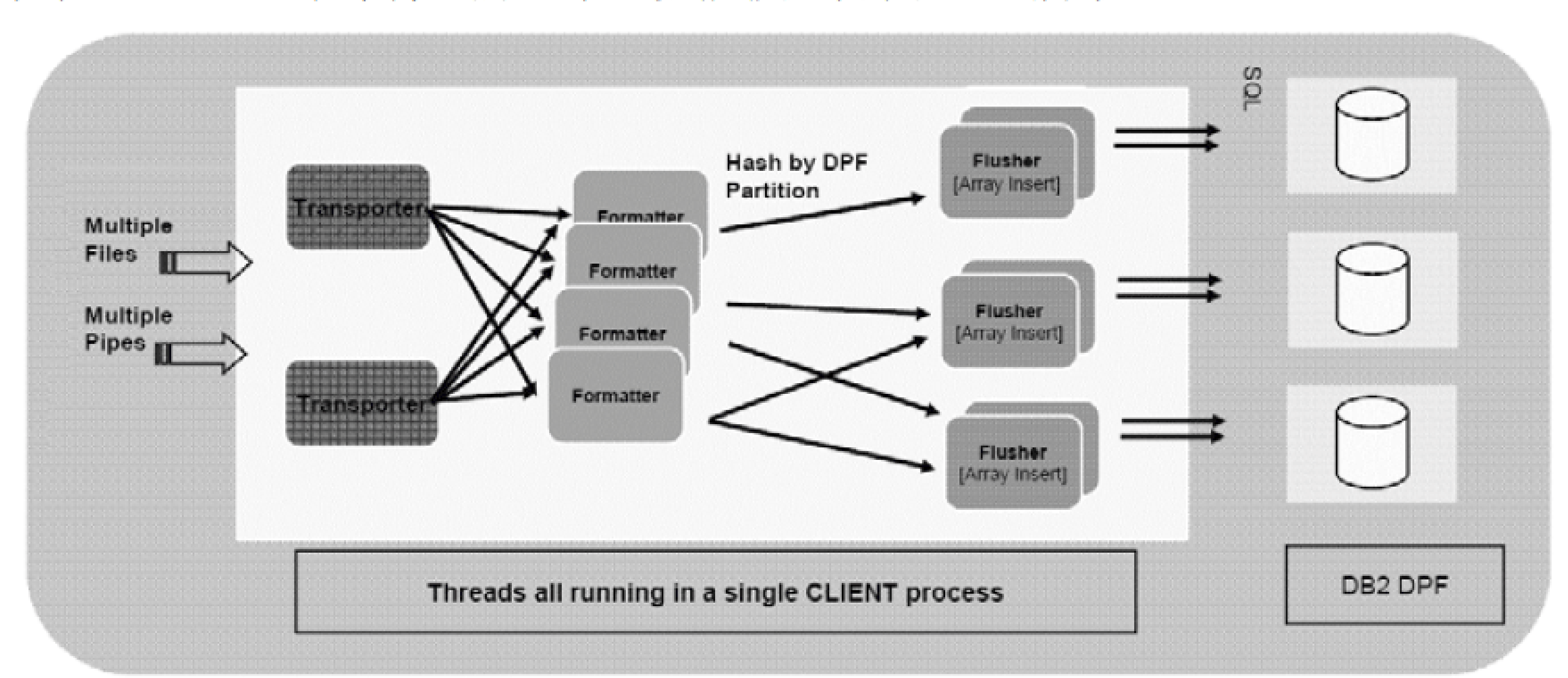


图 13-5　CDI 执行插入过程

(1) 传输：传输程序从数据源中读取，并将记录放置在格式化程序队列中。对于 INSERT 操作，每个输入源都有一个传输程序线程(例如，每个输入文件都有一个线程)。对于 UPDATE 和 DELETE 操作，只有一个传输程序线程。

(2) 格式化：格式化程序会解析每条记录，将数据转换成 DB2 数据库系统所需要的格式并将每条已格式化的记录放置在该记录分区上的其中某个清空程序队列中。格式化程序线程数由 num_formatters 配置参数指定，默认值为(逻辑 CPU 数)/2。

(3) 清空：清空程序发出 SQL 语句以对 DB2 表执行操作。每个分区的清空程序数由 num_flushers_per_partition 配置参数指定，默认值是 1 与(逻辑 CPU 数)/2/分区数之中的较大者。

INGEST 工具有以下相关命令：

```
使用 INGEST 进行数据导入时，最简单的语法结构是：
INGEST-+------+------------------->
  '-DATA-'
 >---| from FILE 或 PIPE | ----------------->
```

```
>---| format 子句 |---------------->
>---| options |---------------------->
```

- INGEST LIST 用来显示当前的 INGEST 进程信息。
- INGEST GET STATS 用来得到当前的 INGEST 状态统计信息。
- INGEST SET 用来设置 INGEST 的一些参数，例如下面一些参数：
  - ◊ COMMIT_COUNT 用来设置将多少条记录作为事务提交。
  - ◊ COMMIT_PERIOD 用来设置多长时间提交一次。
  - ◊ NUM_FLUSHERS_PER_PARTITION 为每个数据库分区的 flusher 数量。
  - ◊ NUM_FORMATTERS。
  - ◊ PIPE_TIMEOUT 为管道输入类型的最大等待接收数据时间。
  - ◊ RETRY_COUNT 为重新尝试的次数。
  - ◊ RETRY_PERIOD 为重新尝试的时间间隔。
  - ◊ SHM_MAX_SIZE 为最大的共享内存。

INGEST 与 LOAD、IMPORT 在功能上的异同

- INGEST 只支持 DEL 和 ASC 数据格式，而 IMPORT 还可以支持 WSD 和 IXF 格式，LOAD 还可以支持 IXF 和 CURSOR 格式。
- INGEST 用于支持 ASC 数据格式的参数与 LOAD、IMPORT 不同，IMPORT、LOAD 使用 METHOD L()来统一设置所有字段的位置，而 INGEST 分别使用 POSITION(:) 来处理每个字段的位置。
- INGEST 支持把错误数据放入文件或表中，LOAD 支持把错误数据放入文件中，IMPORT 不支持把错误数据放到数据表文件中。

与 IMPORT 导入结果相同

对于任何输入数据文件，只要满足以下条件，INGEST 和 IMPORT 命令产生的结果就应该相同：

- 字段定义指定的数据类型和长度与表中的列类型一样。
- ASCII 字符串有效不越界。
- 输入数据使用的字符分割符是一致的。如果使用 ENCLOSED BY '|'，字符字段数据应使用一样的分割符|。

数字类型错误处理

- 对于数字类型的数据，如果其中出现字符串或数据越界现象，INGEST 的处理方式与 LOAD、IMPORT 有所不同。INGEST 会拒绝插入并报错，而 IMPORT、LOAD 则有以下处理方式：
  - ◊ 如果列可空，IMPORT、LOAD 插入 NULL 并报警。
  - ◊ 如果列不可空，LOAD、IMPORT 拒绝插入并报错；
- 除了 DECIMAL 数据类型外，当数据越界时，列可空，IMPORT、LOAD 插入 NULL 并报警；如果不可空，IMPORT、LOAD 拒绝插入并报错。
- 对于 DECIMAL 数据类型，当数据越界时，IMPORT、LOAD 拒绝插入并报错。

CODEPAGE 转换的特点

INGEST 考虑 3 种类型的 CODEPAGE：一个是 DB2 客户端应用程序的 CODEPAGE；另一个是与 INGEST 参数 INPUT CODEPAGE 相关的输入数据的 CODEPAGE，如果没有指定 INPUT CODEPAGE 参数，输入数据默认是应用程序的 CODEPAGE；最后一个是与 create database 命令相关的数据库的 CODEPAGE。

如果字段和表列都指定了 FOR BIT DATA，那么 INGEST 就不做 CODEPAGE 转换。如果表列中有 FOR BIT DATA 定义，那么在进行 INGEST 操作时也在相应的字段使用 FOR BIT DATA，这样可以避免数据的不可预测性。

下面举例说明 INGEST 中的 CODEPAGE 转换。输入数据 CODEPAGE 是 819，应用程序 CODEPAGE 是 850，数据库 CODEPAGE 是 1208， 输入数据是"é"。"é"在 819 中是 X'E9'，在 850 中是 X'82' ，在 1208 中是 X'C3A9'。如果字段和表列都指定为 CHAR FOR BIT DATA，那么存入 DB2 数据库中的数据为 X'E9'; 如果字段和表列都指定为 CHAR，那么存入 DB2 数据库中的数据则为 X'C3A9'。

13.2.2 CDI 实际操作案例

1. 测试场景

- 从单分区数据库直接到多分区数据库的端对端测试。
- 多个 CDI 同时对一张表进行加载的测试。
- 多个 CDI 对多表同时进行加载的测试。
- 和单独的 LOAD 比较一下性能差别。
- CDI 的 MERGE 操作的性能。

2. 测试环境

4 台刀片的 POWER7 机器，配置都相同。3 台作为 DB2 服务器，一台作为热备服务器，同时允许 CDI 客户端在这个备机上。4 台服务器之间用千兆网连接。具体架构如图 13-6 所示。

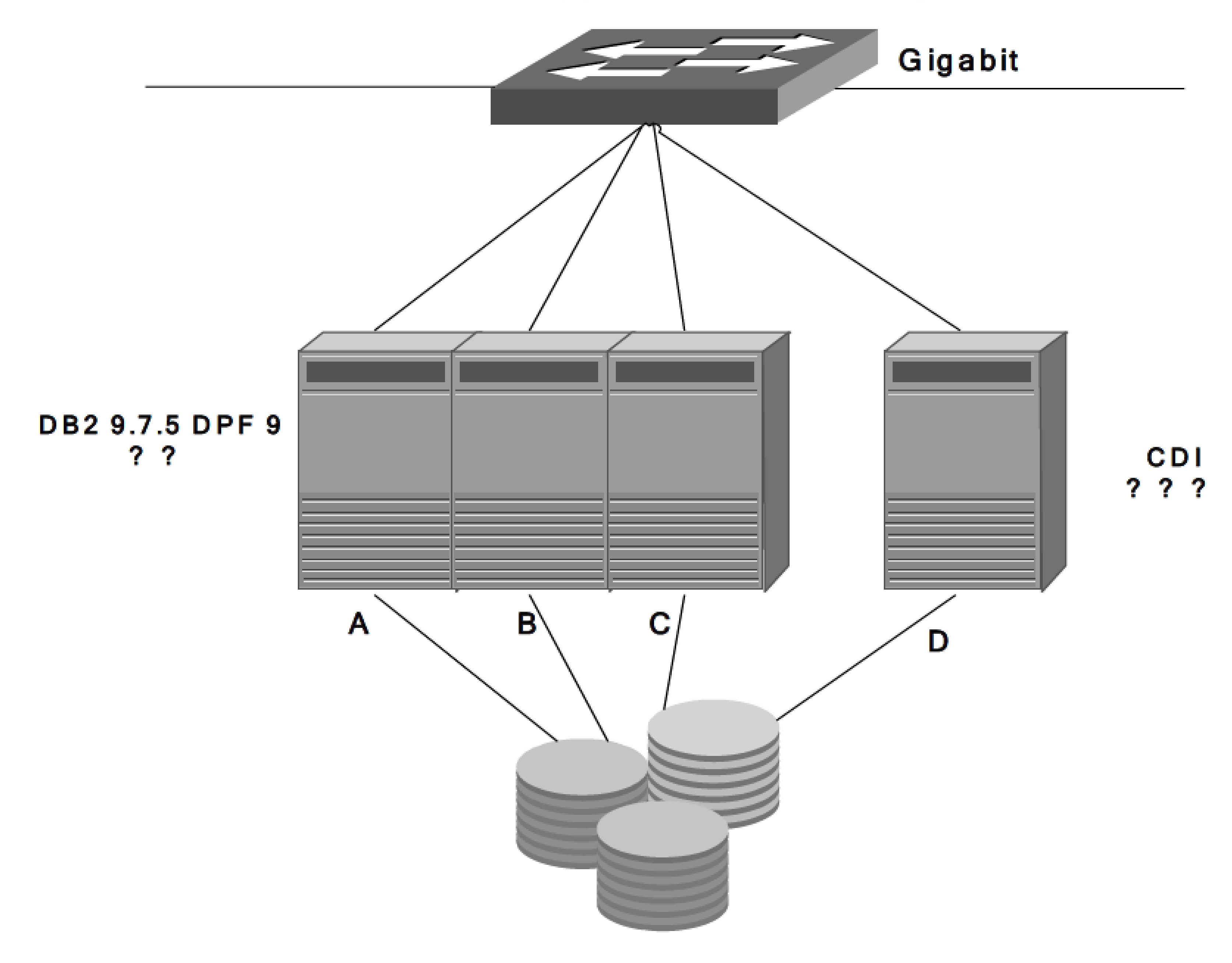


图 13-6　CDI 测试架构图

3. 系统配置

系统配置如表 13-2 所示。

表 13-2　系统配置

数据库服务器	3 power7
数据库客户端	1 power7
操作系统	AIX 6.1
处理器	16 核，3000 MHz
内存	128GB
存储	EMC DMX4
网络	1Gbps
软件	DB2 Server 9.7, DB2 Galileo Client(CDI)

4. 数据库配置

- 数据库设置

数据库 TESTDPF 是 9 分区的数据库。其中，主机 A 上有一个分区 0，存储单分区表；主机 B 和主机 C 各有 4 个分区，存储多分区表。

- 测试表空间定义

测试使用的表空间为如下两个，其中 TEST_DAT 用于存储数据，TEST_IDX 用于存储索引数据。

表空间的创建语句如下：

```
CREATE LARGE TABLESPACE "TEST_DAT" IN DATABASE PARTITION GROUP PDPG PAGESIZE
32768 MANAGED BY DATABASE
USING(FILE '/db2/db2inst1/db2data0/TEST_DAT' 1474560)ON DBPARTITIONNUMS(0)
USING(FILE '/db2/db2inst1/db2data1/TEST_DAT' 1474560)ON
DBPARTITIONNUMS(1)USING(FILE '/db2/db2inst1/db2data2/TEST_DAT' 1474560)ON
DBPARTITIONNUMS(2)
USING(FILE '/db2/db2inst1/db2data3/TEST_DAT' 1474560)ON DBPARTITIONNUMS(3)
USING(FILE '/db2/db2inst1/db2data4/TEST_DAT' 1474560)ON DBPARTITIONNUMS(4)
USING(FILE '/db2/db2inst1/db2data5/TEST_DAT' 1474560)ON DBPARTITIONNUMS(5)
USING(FILE '/db2/db2inst1/db2data6/TEST_DAT' 1474560)ON DBPARTITIONNUMS(6)
USING(FILE '/db2/db2inst1/db2data7/TEST_DAT' 1474560)ON DBPARTITIONNUMS(7)
USING(FILE '/db2/db2inst1/db2data8/TEST_DAT' 1474560)ON DBPARTITIONNUMS(8)
  EXTENTSIZE 32
  PREFETCHSIZE AUTOMATIC
  BUFFERPOOL BP_32K
  OVERHEAD 7.500000
  TRANSFERRATE 0.060000
  NO FILE SYSTEM CACHING
  DROPPED TABLE RECOVERY ON;

CREATE LARGE TABLESPACE "TEST_IDX" IN DATABASE PARTITION GROUP PDPG PAGESIZE
32768 MANAGED BY DATABASE
  USING(FILE '/db2/db2inst1/db2data0/TEST_IDX' 655360)ON DBPARTITIONNUMS(0)
  USING(FILE '/db2/db2inst1/db2data1/TEST_IDX' 655360)ON DBPARTITIONNUMS(1)
  USING(FILE '/db2/db2inst1/db2data2/TEST_IDX' 655360)ON DBPARTITIONNUMS(2)
  USING(FILE '/db2/db2inst1/db2data3/TEST_IDX' 655360)ON DBPARTITIONNUMS(3)
  USING(FILE '/db2/db2inst1/db2data4/TEST_IDX' 655360)ON DBPARTITIONNUMS(4)
  USING(FILE '/db2/db2inst1/db2data5/TEST_IDX' 655360)ON DBPARTITIONNUMS(5)
USING(FILE '/db2/db2inst1/db2data6/TEST_IDX' 655360)ON DBPARTITIONNUMS(6)
USING(FILE '/db2/db2inst1/db2data7/TEST_IDX' 655360)ON DBPARTITIONNUMS(7)
USING(FILE '/db2/db2inst1/db2data8/TEST_IDX' 655360)ON DBPARTITIONNUMS(8)
  EXTENTSIZE 32
```

```
PREFETCHSIZE AUTOMATIC
BUFFERPOOL BP_32K
OVERHEAD 7.500000
TRANSFERRATE 0.060000
NO FILE SYSTEM CACHING
DROPPED TABLE RECOVERY ON;
```

- 缓冲池设置

数据表空间使用 BP_32K 缓冲池。为 32KB 大小的页，共 3.2GB 大小：

```
CREATE BUFFERPOOL "BP_32K"  SIZE 100000 PAGESIZE 32768;
```

5. 测试场景

多个 CDI 同时往一张表里装载数据。本次测试 5 个 CDI 同时往一张表里装载数据的情况，每个 CDI 装载的数据量为 6.25GB，参见表 13-3。

表 13-3　测试场景 1

测试用例	多个 CDI 同时往一张表里装载数据	数据量	共 5 个文件，每个大约 6.25GB 左右，共 32GB
测试用例描述			
功能描述	测试多个 CDI 并发地往同一张表里装载数据，数据能够成功插入到目标表中		
测试步骤	查看表结构，根据表结构编写控制脚本 根据实际需求设置 CDI 的参数(比如多少条记录做一次提交等) 运行 CDI，运行方式：db2 –tvf 脚本名 检查插入表中的数据以验证正确性		
控制脚本	ingest set num_formatters 16; ingest set commit_count 2000; ingest set commit_period 0; ingest set shm_max_size 4 GB; ingest set msg_buf_count 64; ingest set num_flushers_per_partition 8; ingest from file /db2home/tmp/TEMP_SAPDM_TRANS_BAL_CHANGE1.txt <== 不同 CDI 的脚本，所读文件不同 format delimited by '|' input codepage 1208 messages /home/cditest/test/ms.txt restart off insert into TESTUSER.M_DEP_DM_TRANS_INFO;		

(续表)

测试用例	多个 CDI 同时往一个表里装载数据	数据量	共 5 个文件，每个大约 6.25GB 左右，共 32GB
预期结果	多个 CDI 成功将数据装载进目标表，无挂住现象，不需要人工干预		
测试结果	成功		
备注			

多个 CDI 对多张表同时进行加载的测试，参见表 13-4。

表 13-4 测试场景 2

测试用例	多个 CDI 对多张表同时加载	数据量	共 5 个文件，每个大约 6.25GB 左右，共 32GB
测试用例描述			
功能描述	同时启动多个 CDI，每个 CDI 对一个目标表做数据装载		
测试步骤	(1) 查看表结构，根据表结构编写控制脚本 (2) 根据实际需求设置 CDI 的参数(比如多少条记录做一次提交等) (3) 运行 CDI。运行方式为：db2 - tvf 脚本名 (4) 检查插入表中的数据以验证正确性		
控制脚本	ingest set num_formatters 16; ingest set commit_count 2000; ingest set commit_period 0; ingest set shm_max_size 4 GB; ingest set msg_buf_count 64; ingest set num_flushers_per_partition 8; ingest from file /db2home/tmp/TEMP_SAPDM_TRANS_BAL_CHANGE1.txt <== 不同 CDI 的脚本，所读文件不同 format delimited by '|' input codepage 1208 messages /home/cditest/test/ms.txt restart off insert into TESTUSER.M_DEP_DM_TRANS_INFO; <== 不同 CDI 的脚本，目标表不同		
预期结果	不同的 CDI 能将自己所读数据装载到自己的目标表中		
测试结果	成功		
备注			

从单分区数据库直接到多分区数据库的端到端测试，参见表 13-5。

表 13-5　测试场景 3

测试用例	从单分区数据库直接到多分区数据库的端到端测试	数据量	6.25GB，共 15156061 条记录，此数据已经在单分区数据库的表中
测试用例描述			
功能描述	HPU 将单分区表中的数据读出，写入命名管道，CDI 从命名管道读数据并插入到分区表中		
测试步骤	(1) 查看表结构，根据表结构编写控制脚本 (2) 根据实际需求设置 CDI 的参数(比如多少条记录做一次提交等) (3) 启动 HPU 并将记录写入到管道中 (4) 运行 CDI，从管道中读取数据。运行方式为：db2－tvf 脚本名 (5) 检查插入表中的数据以验证正确性		
控制脚本	ingest set num_formatters 32; ingest set commit_count 2000; ingest set commit_period 0; ingest set shm_max_size 4 GB; ingest set msg_buf_count 64; ingest set num_flushers_per_partition 8; ingest from pipe /home/db2inst2/pipe format delimited input codepage 1208 messages /home/cditest/test/ms.txt restart off insert into TESTUSER.M_DEP_DM_TRANS_INFO;		
预期结果	将数据从单分区数据库中导出，插入到目标分区数据库中		
测试结果	成功		
备注			

CDI 的 MERGE 操作，见表 13-6。

表 13-6　测试场景 4

测试用例	CDI 的 MERGE 操作	数据量	1.7GB，共 4 百万条记录
测试用例描述			
功能描述	将数据装载到指定的目标表中。根据表的主键，如果记录已经存在，执行 UPDATE 操作；如果记录不存在，执行 INSERT 操作并测试所有的时间		
测试步骤	(1) 查看表结构，根据表结构编写控制脚本 (2) 根据实际需求设置 CDI 的参数(比如多少条记录做一次提交等) (3) 运行 CDI。运行方式为：db2 - tvf 脚本名 (4) 检查插入表中的数据以验证正确性		
控制脚本	ingest set num_formatters 32; ingest set commit_count 2000; ingest set commit_period 0; ingest set shm_max_size 4 GB; ingest set msg_buf_count 64; ingest set num_flushers_per_partition 8; -- ingest set disk_write no; -- ingest from file /home/cditest/test/t1.txt -- ingest from file /db2home/tmp/xaaa ingest from file /db2home/tmp/M_CRD_DEBT_TRANS_DTL_TEST_ZBW.del -- ingest from file /db2home/tmp/merge.del format delimited -- input codepage 1208 ($field1 char(96), $field2 char(24), $field3 char(30), $field4 char(12), $field5 char(255), $field6 char(75), $field7 char(96), $field8 char(15), $field9 char(9), $field10 char(45), $field11 char(105),		

(续表)

测试用例	CDI 的 MERGE 操作	数据量	1.7GB，共 4 百万条记录
控制脚本	$field12 char(18), $field13 char(9), $field14 char(9), $field15 char(60), $field16 char(3), $field17 char(3), $field18 decimal(31,9)external, $field19 char(24), $field20 char(96), $field21 char(3), $field22 decimal(31,9)external, $field23 char(3), $field24 char(3), $field25 char(24), $field26 char(18), $field27 char(48), $field28 char(8), $field29 char(8), $field30 char(8)) messages /home/cditest/test/ms.txt restart off merge into TESTUSER.M_CRD_DEBT_TRANS_DTL_TEST_ZBW on TRANS_ID=$field1 and TRANS_DT=$field2 and TRANS_CHANNEL=$field3 and TRANS_TYPE=$field4 when matched then update set(CARD_ID,CARD_NUM ,DESCS,CURR_CD,BANK_COUNTRY,BANK_CODE,ACCT_NUM,TRANS_TIME,TRANS_COUNTRY,TRANS_REGION, MERCHANT_NAME,SAVE_DRAW_TYPE,DR_CR_FLAG,TRANS_AMT,ORIGINAL_TRANS_DT,ORIGINAL_TRANS_ID, ORIGINAL_TRANS_TYPE,ORIGINAL_TRANS_AMT,REVERSAL_FLAG,STAT,LAST_UPDATE_DT,LAST_UPDATE_TIME,		

(续表)

测试用例	CDI 的 MERGE 操作	数据量	1.7GB，共 4 百万条记录
控制脚本	LAST_UPDATE_USER,BUSS_DT,DATA_DT,ADD_DT)=($field5,$field6,$field7, $field8, $field9,$field10, $field11, $field12,$field13, $field14,$field15,$field16,$field17,$field18, $field19,$field20,$field21,$field22,$field23,$field24,$field25,$field26, $field27,$field28,$field29,$field30) when not matched then insert values($field1, $field2,$field3,$field4, $field5,$field6,$field7, $field8, $field9,$field10, $field11, $field12,$field13, $field14,$field15,$field16,$field17,$field18, $field19,$field20,$field21,$field22,$field23,$field24,$field25,$field26, $field27,$field28,$field29,$field30);		
预期结果	数据的 MERGE 操作成功		
测试结果	数据的 MERGE 操作成功，使用时间是 6 分 19.98 秒		
备注	使用 IMPORT 的 INSERT_UPDATE 来实现，速度非常慢。测试的结果是分区表要用 103 分钟，非分区表要用 30 多分钟		

CDI 装载(INSERT 操作)数据到一张表中的性能，见表 13-7。

表 13-7　测试场景 5

测试用例	CDI 装载 (INSERT 操作) 数据到一张表中	数据量	数据大小为 31.27GB,共 75780307 条记录
测试用例描述			
功能描述	CDI 往同一目标表里装载数据，数据能够成功插入到目标表中并测试性能		
测试步骤	(1) 查看表结构，根据表结构编写控制脚本 (2) 根据实际需求设置 CDI 的参数(比如多少条记录做一次提交等) (3) 运行 CDI。运行方式为：db2 - tvf 脚本名 (4) 检查插入表中的数据以验证正确性，并记录所用时间		

(续表)

测试用例	CDI 装载(INSERT 操作)数据到一张表中	数据量	数据大小为 31.27GB,共 75780307 条记录
控制脚本	ingest set num_formatters 32; ingest set commit_count 2000; ingest set commit_period 0; ingest set shm_max_size 4 GB; ingest set msg_buf_count 64; ingest set num_flushers_per_partition 8; ingest from file /db2home/tmp/TEMP_SAPDM_TRANS_BAL_CHANGE.txt format delimited by '|' input codepage 1208 messages /home/cditest/test/ms.txt restart off insert into TESTUSER.M_DEP_DM_TRANS_INFO;		
预期结果	CDI 将数据插入到目标表中，并尽可能使用最短的时间		
测试结果	所用时间为 6 分 1.19 秒。		
备注	速度达到 88.7MB/s，此时网络达到 113MB/S~115MB/s(网络达到了瓶颈)，此速度能远远满足业务的要求。LOAD 为做任何优化所用时间是 6 分 46.17 秒，都是默认参数。如果做一些优化，性能应该有所提升		

13.3　缩骨大法——自适应压缩

13.3.1　基本介绍

“缩骨大法者，全身骨骼伸缩自如，可缩于一团，轻松入瓮，出瓮后，迎风一展，恢复本身，即使现代瑜伽，也自叹不如”。在数据库领域，DB2 V9 以后引入表压缩，DB2 V9.7 以后引入索引压缩，在 DB2 V10.1 以后更是引入了一种先进的关于表的行压缩技术——自适应压缩技术。为了以示区别，DB2 V10.1 之前的行压缩技术又被称为经典行压缩或静态压缩。那么自适应压缩和经典行压缩的区别在哪里呢？

- 经典行压缩：经典行压缩(有时又称为静态压缩或深度压缩)通过将行间重复的值的模式替换为较短符号字符串来压缩数据行。
- 自适应压缩：自适应压缩通过将在单个数据页内行间重复值的模式替换为较短符号字符串来压缩数据行，可以较大幅度提高压缩率。

13.3.2 自适应压缩的工作方式

自适应压缩实际使用两种压缩方法。

第一种方法使用表级别压缩字典，根据表中数据抽样的重复情况来整体压缩数据，第一次创建表级别压缩字典时，系统使用启用经典行压缩后添加至表中的大约第一个兆字节数据中的数据样本构建该字典。除非显式导致字典重建，否则系统不会再次更新该字典。即使重建该字典，该字典也只反映整个表中数据的样本。也就是说，表级别字典是静态字典，表级别压缩字典存储在应用该字典的对象的隐藏行中，并且会高速缓存在内存中以便快速访问。此字典不会占用太多空间。即使对于极大的表，压缩通常也只占用大约 100 KB 空间。

第二种方法使用基于页级别字典的压缩算法以根据每个数据页中的数据重复情况来压缩数据，这些字典将重复字节模式映射至小得多的符号，然后这些符号会替换表中的较长字节模式。表级别压缩字典存储在为其创建该字典的表对象中，用于压缩整个表的数据。页级别压缩字典与数据页中的数据存储在一起，仅用于压缩该页中的数据，需要的空间也极少。

使用自适应压缩技术的另外一个好处是，无须执行显式表重组即可保持这些高压缩率。页面级压缩字典是自动创建的，如果页面的内容发生重大变化，页面级压缩字典会自动重新创建。当页面装满时，会应用页面级压缩，这样会迅速释放页面上的更多存储空间。

13.3.3 启用或禁用自适应压缩

自适应压缩可在 Enterprise Server Edition 的存储优化功能(SOF)中找到。另外，SOF 包含在 Advanced Enterprise Server Edition 中。

在 DB2 V10.1 以后，自适应压缩是启用行压缩的默认行为。通过在 CREATE TABLE 中使用 COMPRESS YES STATIC 子句，可继续在 DB2 V10.1 中使用经典行压缩：

```
>>-CREATE TABLE--table-name----------------------------->

      .-COMPRESS NO---------------.
>--●--+---------------------------+--●--+--------------------+--->
      |              .-ADAPTIVE-.    |     '-VALUE COMPRESSION-'
      '-COMPRESS YES-+----------+-'
```

```
'-STATIC---'
```

为现有的表启用自适应压缩，只需要使用 ALTER TABLE 语句的 COMPRESS YES 子句或新的 COMPRESS YES ADAPTIVE 子句：

```
>>-ALTER TABLE--table-name------------------------------------->

      .-COMPRESS NO---------------.
>--●--+---------------------------+--●--+------------------+--->
      |             .-ADAPTIVE-.  |     '-VALUE COMPRESSION-'
      '-COMPRESS YES-+----------+-'
'-STATIC---'
```

为现有表启用自适应压缩之后，所有后续更新或新添加的数据都会进行自适应压缩。表中已存在的数据如果要应用自适应压缩，需要执行一次表重组，重建压缩字典。

要对表禁用压缩，使用带有 COMPRESS NO 选项的 ALTER TABLE 语句，以后添加的行不会被压缩，现有行将保持已压缩状态。要使禁用压缩后的整个表都不压缩，必须使用 REORG TABLE 命令执行表重组。

13.3.4 评估表压缩率

对于 DB2 V9.7，评估压缩率的表函数为 ADMIN_GET_TAB_COMPRESS_INFO 和 ADMIN_GET_TAB_COMPRESS_INFO_V97(推荐使用)。

在 DB2 V9.7 中，ADMIN_GET_TAB_COMPRESS_INFO 的语法如下：

```
>>-ADMIN GET TAB COMPRESS INFO--(--tabschema--,--tabname--,--execmode--)-><

其中，excemode 参数的值：
'REPORT'--Reports compression information as of last generation. This is the
          default value.
'ESTIMATE'--Generates new compression information based on the current table.
```

在 DB2 V9.7 中，ADMIN_GET_TAB_COMPRESS_INFO_V97 语法如下：

```
>>-ADMIN_GET_TAB_COMPRESS_INFO_V97--(--tabschema--,--tabname--,--execmode--
)-><

其中，excemode 参数的值：
'REPORT'--Reports compression information as of last generation. This is the
          default value.
'ESTIMATE'--Generates new compression information based on the current table.
```

对于 DB2 V10.1，评估压缩率的表函数为 ADMIN_GET_TAB_COMPRESS_INFO。虽

然在 DB2 V9.7 中和 V10.1 中都有表函数 ADMIN_GET_TAB_COMPRESS_INFO，但是它们的语法却不同，输出的列也不同，这一点请读者留意。

DB2 V10.1 中的 ADMIN_GET_TAB_COMPRESS_INFO 表函数可以同时评估经典行压缩、自适应压缩以及目前的压缩率。由于该表函数评估的准确程度取决于统计信息，因此在评估前最好用 runstats 更新一下最新的统计信息，语法如下：

```
>>-ADMIN_GET_TAB_COMPRESS_INFO--(--tabschema--,--tabname--)----><
```

示例如下：

```
--下面的表 SAPSR3.BCA_CN_LINK_V10 为分区表，输出的压缩率为各个数据分区的压缩率

db2 "select substr(TABNAME, 1, 30)TABNAME, PCTPAGESSAVED_CURRENT,
PCTPAGESSAVED_STATIC, PCTPAGESSAVED_ADAPTIVE from
table(sysproc.admin_get_tab_compress_info('SAPSR3','BCA_CN_LINK'))as t "

TABNAME     PCTPAGESSAVED_CURRENT PCTPAGESSAVED_STATIC PCTPAGESSAVED_ADAPTIVE
------------------------------------------------------------------------
BCA_CN_LINK            0                  0                   0
BCA_CN_LINK           70                 67                  78
BCA_CN_LINK           71                 67                  78
BCA_CN_LINK           71                 67                  78
BCA_CN_LINK           70                 67                  78
BCA_CN_LINK           71                 67                  78
BCA_CN_LINK           66                 67                  78
BCA_CN_LINK           71                 68                  78
BCA_CN_LINK           70                 68                  78
BCA_CN_LINK           71                 67                  78
BCA_CN_LINK           69                 67                  78
BCA_CN_LINK           70                 67                  78
BCA_CN_LINK           69                 67                  78
BCA_CN_LINK           70                 67                  78
BCA_CN_LINK           70                 67                  78
BCA_CN_LINK           70                 67                  77
BCA_CN_LINK           70                 67                  77
```

```
   BCA_CN_LINK             0                 0                   0


     18 record(s)selected.
```

13.3.5　经典行压缩和自适应压缩的对比测试

测试目标：为了更好地认识自适应压缩的优点所在，接下来在 DB2 V10.1 中进行经典行压缩和自适应压缩的对比测试。

测试设备：IBM P780、AIX 6.1、DB2 V10.1。

```
$db2level
DB21085I  Instance "db2v10" uses "64" bits and DB2 code release "SQL10010" with
level identifier "0201010E".
Informational tokens are "DB2 v10.1.0.0", "s120403", "AIX64101", and Fix Pack "0".
Product is installed at "/software/V10".
```

测试思路：针对表 A，按照 A 的语法定义新建两个一模一样的表 B 和 C，其中 B 采用经典行压缩，C 采用自适应压缩。通过 load cursor 脚本从 A 分别导入数据到 B 和 C 中，测试 B 和 C 压缩率的区别。

测试过程：

(1) 在 DB2 V10.1 中，针对已有表 BCA_CN_LINK(该表为分区表)，新建表 BCA_CN_LINK_V97 和 BCA_CN_LINK_V10，这两个表也均为分区表，其中 BCA_CN_LINK_V97 的压缩属性为 static，BCA_CN_LINK_V10 的压缩属性为 adaptive。这两张表一开始均为空表。

```
   其中，BCA_CN_LINK_V97 的 DDL 如下，仅供参考
   ------------------------------------------------
   -- DDL Statements for table "SAPSR3  "."BCA_CN_LINK_V97"
   ------------------------------------------------


   CREATE TABLE "SAPSR3  "."BCA_CN_LINK_V97" (
   "CLIENT" VARCHAR(9)NOT NULL WITH DEFAULT '000' ,
   "CN_LINK_ID" CHAR(16)FOR BIT DATA NOT NULL ,
   "CHANGE_TSTAMP" DECIMAL(21,7)NOT NULL WITH DEFAULT 0 ,
   "CONTRACT_INT" CHAR(16)FOR BIT DATA ,
   "OBJECT_TYP" VARCHAR(12)NOT NULL WITH DEFAULT ' ' ,
```

```
"OBJECT_ID" CHAR(16)FOR BIT DATA ,
"FUNCTION" VARCHAR(18)NOT NULL WITH DEFAULT ' ' ,
"VALID_FROM" DECIMAL(15,0)NOT NULL WITH DEFAULT 0 ,
"VALID_TO" DECIMAL(15,0)NOT NULL WITH DEFAULT 0 ,
"VALID_TO_REAL" DECIMAL(15,0)NOT NULL WITH DEFAULT 0 ,
"FLG_TOBE_REL" VARCHAR(3)NOT NULL WITH DEFAULT ' ' ,
"FLG_OWNER" VARCHAR(3)NOT NULL WITH DEFAULT ' ' ,
"CHANGE_USER" VARCHAR(36)NOT NULL WITH DEFAULT ' ' ,
"CHANGE_BTCATG" VARCHAR(18)NOT NULL WITH DEFAULT ' ' ,
"FLG_FIRST_REMAIN" VARCHAR(3)NOT NULL WITH DEFAULT ' ' )
COMPRESS YES STATIC
VALUE COMPRESSION
INDEX IN "PR2#CNLK_IG" PARTITION BY RANGE("CLIENT", "CONTRACT_INT")
(
PART BCA_CNLK_0
STARTING(710, x'00000000000000000000000000000000')
IN PR2#CNLK_D0
INDEX IN PR2#CNLK_I0
LONG IN PR2#CNLK_D0,
PART BCA_CNLK_1
STARTING(710, x'10000000000000000000000000000000')
IN PR2#CNLK_D1
INDEX IN PR2#CNLK_I1
LONG IN PR2#CNLK_D1,
PART BCA_CNLK_2
STARTING(710, x'20000000000000000000000000000000')
IN PR2#CNLK_D2
INDEX IN PR2#CNLK_I2
LONG IN PR2#CNLK_D2,
PART BCA_CNLK_3
STARTING(710, x'30000000000000000000000000000000')
IN PR2#CNLK_D3
INDEX IN PR2#CNLK_I3
```

```
LONG IN PR2#CNLK_D3,
PART BCA_CNLK_4
STARTING(710, x'40000000000000000000000000000000')
IN PR2#CNLK_D4
INDEX IN PR2#CNLK_I4
LONG IN PR2#CNLK_D4,
PART BCA_CNLK_5
STARTING(710, x'50000000000000000000000000000000')
IN PR2#CNLK_D5
INDEX IN PR2#CNLK_I5
LONG IN PR2#CNLK_D5,
PART BCA_CNLK_6
STARTING(710, x'60000000000000000000000000000000')
IN PR2#CNLK_D6
INDEX IN PR2#CNLK_I6
LONG IN PR2#CNLK_D6,
PART BCA_CNLK_7
STARTING(710, x'70000000000000000000000000000000')
IN PR2#CNLK_D7
INDEX IN PR2#CNLK_I7
LONG IN PR2#CNLK_D7,
PART BCA_CNLK_8
STARTING(710, x'80000000000000000000000000000000')
IN PR2#CNLK_D8
INDEX IN PR2#CNLK_I8
LONG IN PR2#CNLK_D8,
PART BCA_CNLK_9
STARTING(710, x'90000000000000000000000000000000')
IN PR2#CNLK_D9
INDEX IN PR2#CNLK_I9
LONG IN PR2#CNLK_D9,
PART BCA_CNLK_A
STARTING(710, x'A0000000000000000000000000000000')
```

```
IN PR2#CNLK_DA
INDEX IN PR2#CNLK_IA
LONG IN PR2#CNLK_DA,
PART BCA_CNLK_B
STARTING(710, x'B0000000000000000000000000000000')
IN PR2#CNLK_DB
INDEX IN PR2#CNLK_IB
LONG IN PR2#CNLK_DB,
PART BCA_CNLK_C
STARTING(710, x'C0000000000000000000000000000000')
IN PR2#CNLK_DC
INDEX IN PR2#CNLK_IC
LONG IN PR2#CNLK_DC,
PART BCA_CNLK_D
STARTING(710, x'D0000000000000000000000000000000')
IN PR2#CNLK_DD
INDEX IN PR2#CNLK_ID
LONG IN PR2#CNLK_DD,
PART BCA_CNLK_E
STARTING(710, x'E0000000000000000000000000000000')
IN PR2#CNLK_DE
INDEX IN PR2#CNLK_IE
LONG IN PR2#CNLK_DE,
PART BCA_CNLK_F
STARTING(710, x'F0000000000000000000000000000000')
ENDING(710, x'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF')
IN PR2#CNLK_DF
INDEX IN PR2#CNLK_IF
LONG IN PR2#CNLK_DF,
PART BCA_PAYMITEM_LOW
STARTING(000, MINVALUE)
ENDING(709, MAXVALUE)
IN PR2#CNLKD
```

```
INDEX IN PR2#CNLKI
LONG IN PR2#CNLKD,
PART BCA_PAYMITEM_HIGH
STARTING(711, MINVALUE)
ENDING(999, MAXVALUE)
IN PR2#CNLKD
INDEX IN PR2#CNLKI
LONG IN PR2#CNLKD
);
```

```
$db2 "select count(*)from sapsr3.bca_cn_link_V97"
1
-----------
          0
  1 record(s)selected.
$db2 "select count(*)from sapsr3.bca_cn_link_V10"
1
-----------
          0
  1 record(s)selected.
```

(2) 导入数据，对比 load 时间，采用 load cursor 的方式。

```
load_cursor_BCA_CN_LINK_V97.sql
内容：
connect  to  PNB;
select  current  time  from  syscat.tables  fetch  first  1  rows  only;
DECLARE  mycurs  CURSOR  FOR  SELECT  *  from  SAPSR3.BCA_CN_LINK ;
load FROM mycurs OF cursor INSERT INTO SAPSR3.BCA_CN_LINK_V97 NONRECOVERABLE;
select  current  time  from  syscat.tables  fetch  first  1  rows  only;
terminate;
load 结果：14:55:21-16:44:19，耗时 1 小时 49 分钟，行数 305541952
```

```
load_cursor_BCA_CN_LINK_V10.sql
内容：
connect to PNB;
select current time from syscat.tables fetch first 1 rows only;
DECLARE mycurs CURSOR FOR SELECT * from SAPSR3.BCA CN LINK ;
load FROM mycurs OF cursor INSERT INTO SAPSR3.BCA CN LINK V10 NONRECOVERABLE;
select current time from syscat.tables fetch first 1 rows only;
terminate;
load 结果：16:48:40-18:54:43，耗时 2 小时 6 分钟，行数 305541952
```

通过对比可见，使用自适应压缩的加载时间比普通行压缩的加载时间稍长，这可能是要进行两级压缩的原因。

(3) 评估表的压缩率。

对于经典行压缩，这里采用 admin_get_tab_compress_info_v97 表函数来评估压缩率，目的是为了模拟在 DB2 9.7 下的评估效果。利用这个函数来评估压缩率，耗时 3 分多钟。

```
$time db2 "select substr(TABNAME, 1, 30), COMPRESS_ATTR, DICT_BUILDER,
DICT_BUILD_TIMESTAMP, PAGES_SAVED_PERCENT from
table(sysproc.admin_get_tab_compress_info_v97('SAPSR3','BCA_CN_LINK_V97',
'ESTIMATE'))as t"

输出：
1                              COMPRESS_ATTR DICT_BUILDER
DICT_BUILD_TIMESTAMP       PAGES_SAVED_PERCENT
------------------------------ -------------
---------------------------------------------------------------------------
BCA_CN_LINK_V97 Y  NOT BUILT                  -               0
BCA_CN_LINK_V97 Y  NOT BUILT                  -               0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-25-16.53.09.000000  66
BCA_CN_LINK_V97 Y  NOT BUILT                  -               0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.53.23.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                  -               0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.53.37.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                  -               0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.53.51.000000   66
```

```
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0

BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.54.05.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.54.20.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.54.34.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.54.48.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.55.02.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.55.16.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.55.30.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.55.44.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.55.58.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.56.13.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.56.27.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  TABLE FUNCTION 2012-07-2516.56.41.000000   66
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0
BCA_CN_LINK_V97 Y  NOT BUILT                      -              0

  36 record(s)selected.

real    3m46.57s
user    0m0.02s
sys     0m0.01s
```

对于自适应行压缩，采用 DB2 V10.1 中的 admin_get_tab_compress_info 表函数来评估压缩率。

```
    $time db2 "select substr(TABNAME, 1, 30), PCTPAGESSAVED_CURRENT,
PCTPAGESSAVED_STATIC, PCTPAGESSAVED_ADAPTIVE from
table(sysproc.admin_get_tab_compress_info('SAPSR3','BCA_CN_LINK_V10'))as t "

    1          CTPAGESSAVED_CURRENT PCTPAGESSAVED_STATIC PCTPAGESSAVED_ADAPTIVE
    ------------------------------ -------------------- --------------------
    BCA_CN_LINK_V10        0                 0                   0
    BCA_CN_LINK_V10        70                67                  78
    BCA_CN_LINK_V10        71                67                  78
    BCA_CN_LINK_V10        71                67                  78
    BCA_CN_LINK_V10        70                67                  78
    BCA_CN_LINK_V10        71                67                  78
    BCA_CN_LINK_V10        66                67                  78
    BCA_CN_LINK_V10        71                68                  78
    BCA_CN_LINK_V10        70                68                  78
    BCA_CN_LINK_V10        71                67                  78
    BCA_CN_LINK_V10        69                67                  78
    BCA_CN_LINK_V10        70                67                  78
    BCA_CN_LINK_V10        69                67                  78
    BCA_CN_LINK_V10        70                67                  78
    BCA_CN_LINK_V10        70                67                  78
    BCA_CN_LINK_V10        70                67                  77
    BCA_CN_LINK_V10        70                67                  77
    BCA_CN_LINK_V10         0                 0                   0

      18 record(s)selected.
    real     0m40.85s
    user     0m0.00s
    sys      0m0.01s
```

该表函数耗时 40 多秒，显然相对于 DB2 V9.7 而言，DB2 V10.1 下评估压缩率的表函数的执行效率有了显著提高。

(4) 对表 BCA_CN_LINK_V97 和 BCA_CN_LINK_V10 执行 RUNSTATS，查看表的真实压缩率。

```
    $nohup db2 runstats on table sapsr3.bca_cn_link_v97 with distribution and
detailed indexes all &
    通过查询 SYSCAT.TABLES，发现 BCA_CN_LINK_V97 的真实压缩率为 44%
    $db2 "SELECT SUBSTR(TABNAME,1,20)AS TABNAME, NPAGES, FPAGES, PCTPAGESSAVED,
AVGROWSIZE FROM SYSCAT.TABLES WHERE TABNAME = 'BCA_CN_LINK_V97'"
    TABNAME              NPAGES    FPAGES    PCTPAGESSAVED AVGROWSIZE
    -------- -------------------- -------------------- ------------- ----------
    BCA_CN_LINK_V97   1593799    1593895          44        84
      1 record(s)selected.
    $nohup db2 runstats on table sapsr3.bca_cn_link_v10 with distribution and
detailed indexes all &
    通过查询 SYSCAT.TABLES，发现 BCA_CN_LINK_V97 的真实压缩率为 69%,这里可以看到自适应
压缩的显著优势
    $db2 "SELECT SUBSTR(TABNAME,1,20)AS TABNAME, NPAGES, FPAGES, PCTPAGESSAVED,
AVGROWSIZE FROM SYSCAT.TABLES WHERE TABNAME = 'BCA_CN_LINK_V10'"

    TABNAME          NPAGES      FPAGES    PCTPAGESSAVED AVGROWSIZE
    -------------------- -------------------- -------------------- -----------
    BCA_CN_LINK_V10    866104    866201          69        33
      1 record(s)selected.
```

通过对比，使用自适应压缩的实际压缩率比普通行压缩的压缩率要高许多，这可能也是页级压缩的作用所在。

(5) 前面得到的压缩率效果是 DB2 自己创建的数据字典，那么如果进行 REORG，重建数据字典，能不能看到自适应压缩的另一个好处——“无须执行显式表重组即可保持这些高压缩率”呢？

对 BCA_CN_LINK_V97 和 BCA_CN_LINK_V10 表均执行 REORG 命令，根据现有的数据重建压缩字典，然后在两张表上运行 RUNSTATS 命令。比较两张表的压缩率情况。

```
    $nohup db2 reorg table sapsr3.bca_cn_link_v97  use PSAPTEMP16
RESETDICTIONARY  &
    $nohup db2 runstats on table sapsr3.bca_cn_link_v97 with distribution and
detailed indexes all &
```

```
    $db2 "SELECT SUBSTR(TABNAME,1,20)AS TABNAME, NPAGES, FPAGES, PCTPAGESSAVED,
AVGROWSIZE FROM SYSCAT.TABLES WHERE TABNAME = 'BCA_CN_LINK_V97'"
    TABNAME    NPAGES       FPAGES  PCTPAGESSAVED   AVGROWSIZE
    -------------------- -------------------- -------------------- -----------
    BCA_CN_LINK_V97   957787  957853           66        51
    $nohup db2 reorg table sapsr3.bca_cn_link_v10  use PSAPTEMP16
RESETDICTIONARY  &
    $nohup db2 runstats on table sapsr3.bca_cn_link_v10 with distribution and
detailed indexes all &
    $db2 "SELECT SUBSTR(TABNAME,1,20)AS TABNAME, NPAGES, FPAGES, PCTPAGESSAVED,
AVGROWSIZE FROM SYSCAT.TABLES WHERE TABNAME = 'BCA_CN_LINK_V10'"
    TABNAME       NPAGES  FPAGES   PCTPAGESSAVED AVGROWSIZE
    -------------------- -------------------- -------------------- -----------
    BCA_CN_LINK_V10  659897   659963           77        27
      1 record(s)selected.
```

通过上面的输出，可以看出 REORG 后，经典行压缩的表压缩率提高了 22%(从 44%提高到 66%)，而自适应压缩的表压缩率提高了 8%(从 69%提高到了 77%)。

也就是说，对于 DB2 自己创建的字典来说，显然自适应压缩通过两种方法建立的数据字典更为准确地反映了数据的情况，无须执行显式表重组即可保持这些高压缩率。

(7) 对比测试结果汇总。通过 load cursor 的方法导入数据，静态压缩和自适应压缩的对比如表 13-8 所示。

表 13-8　压缩测试结果

表　　名	压 缩 方 式	load cursor+runstats	load cursor+ reorg+runstats
BCA_CN_LINK_V97	静态压缩	44%	66%
BCA_CN_LINK_V10	自适应压缩	69%	77%

13.3.6　归档日志压缩

这里顺便提一下，在 DB2 V10.1 中，新引入了数据库参数 logarchcompr1，设成 ON 之后，可以对归档日志进行压缩以节省归档文件系统的空间。

测试如下：

```
    db2=> update db cfg for testdb using logarchcompr1 on
```

```
db2=> db2stop
db2=> db2start
```

查看归档日志的对比情况，压缩率为 1-175298517/524296192=1-33.43%=66.57%。

```
开启归档日志压缩前：
-rw-r-----   1 db2pnb  dbpnbadm 524296192 Jul 27 11:58 S0012179.LOG
-rw-r-----   1 db2pnb  dbpnbadm 524296192 Jul 27 12:39 S0012180.LOG
```

```
开启归档日志压缩后，归档日志压缩后每次的大小均不太一样，平均的归档日志压缩率大概在 66%
左右：
-rwx------   1 db2pnb  dbpnbadm 165407425 Jul 27 14:06 S0012182.LOG
-rwx------   1 db2pnb  dbpnbadm 174333702 Jul 27 14:07 S0012183.LOG
-rwx------   1 db2pnb  dbpnbadm 175465946 Jul 27 14:07 S0012184.LOG
-rwx------   1 db2pnb  dbpnbadm 174356119 Jul 27 14:08 S0012185.LOG
-rwx------   1 db2pnb  dbpnbadm 175298517 Jul 27 14:08 S0012186.LOG
-rwx------   1 db2pnb  dbpnbadm 175659075 Jul 27 14:08 S0012187.LOG
-rwx------   1 db2pnb  dbpnbadm 175249831 Jul 27 14:09 S0012188.LOG
```

13.4　乾坤大挪移——灾备功能增强

13.4.1　基本介绍

“东南西北中，天地日月明。阴阳五行法，乾坤大挪移。”数据库的灾备功能就如同乾坤大挪移的主旨，在于颠倒一刚一柔、一阴一阳的乾坤二气。此功夫并非一朝一夕能够练成，必须日积月累，因此从 DB2 V10.1 开始，IBM 又为其灾备功能(DB2 HADR)增加了很多全新的功能。

13.4.2　超级异步

在 DB2 V10.1 之前，HADR 的同步模式只有 SYNC、NEARSYNC 和 ASYNC 三种，但即使是保护力度最小的 ASYNC 模式，对主机事务也是有影响的，尤其对于插入操作频繁的导入操作，性能甚至可能降低几倍。因此为了满足对数据一致性要求不高但是对主机性能要求很高的系统，可以采用 SUPERASYNC 这种模式。

此方式具有最短的事务响应时间，但如果主系统发生故障，那么事务失败的可能性也

最大。当不希望事务由于网络中断或拥塞而受到阻塞或经历较长的响应时间时，此方式很有用。

在此方式中，HADR 对永远不会处于对等状态或断开对等状态。仅当日志记录已写入主数据库的日志文件中时，才会认为日志写入是成功的。因为主系统不会等待来自备用系统的应答，所以当事务仍处于正在传入备用系统的过程中时，可能会认为事务已落实。如图 13-7 所示，主数据库中数据库日志的产生与发送完全分离，二者没有任何依赖。这样一来，HADR 对主数据库业务的影响降到了最低，因此只要 log writer 写入本地日志成功，事务就会提交。

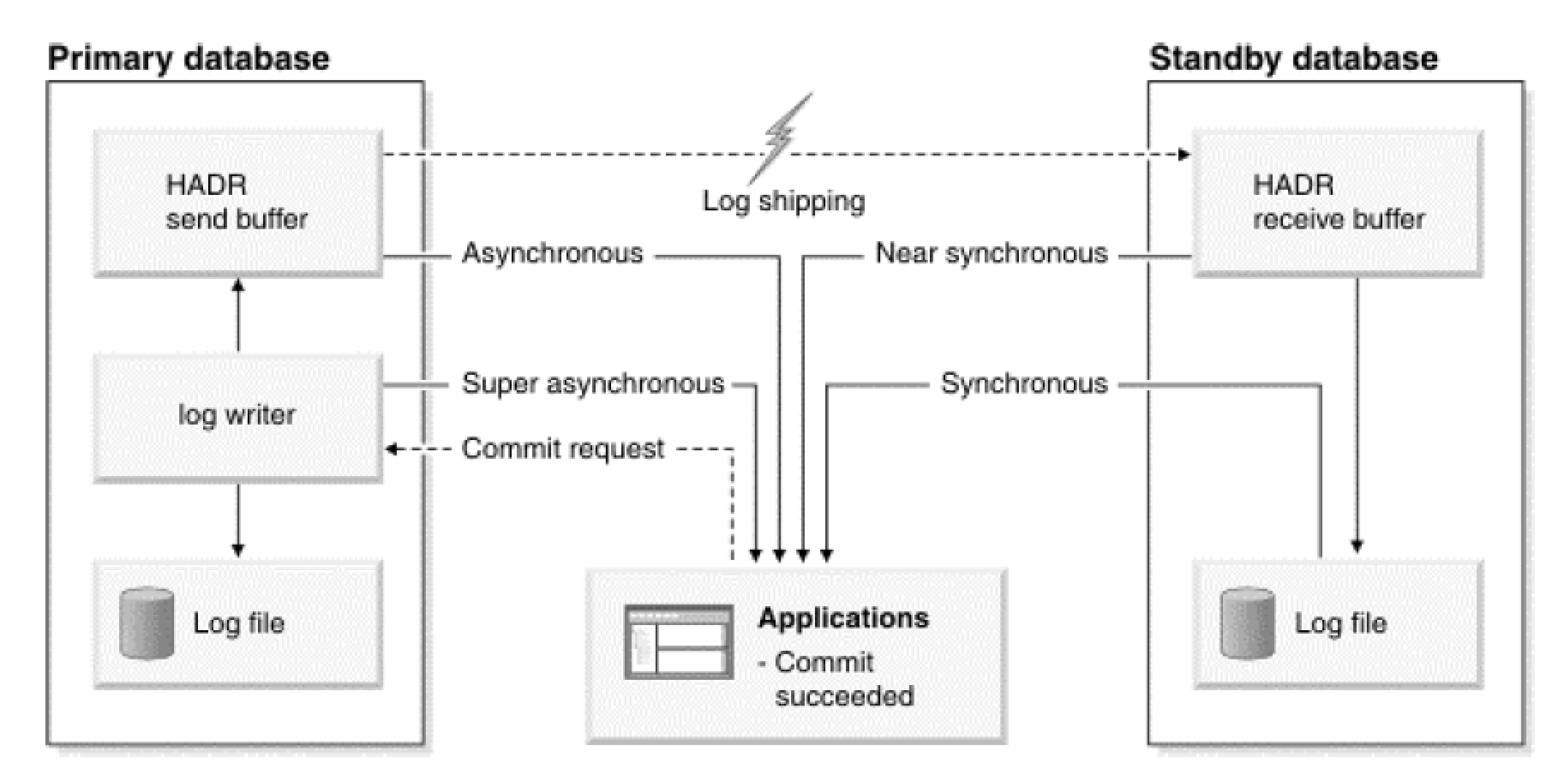


图 13-7　高可用性和灾难恢复(HADR)的同步方式

在 DB2 V10.1 里，用户可以通过修改主备机 db cfg 来配置 SUPERASYNC 模式：

```
$db2 update db cfg for test using HADR_SYNCMODE SUPERASYNC
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
```

配置好之后，可以使用 db2pd -hadr 来检查 HADR 的状态，这个命令相比 DB2 V9.7 有了非常大的变化，示例如下：

```
$db2pd -d test -hadr

Database Member 0 -- Database TEST -- Standby -- Up 0 days 00:00:45 -- Date
09/14/2012 18:20:25

                          HADR_ROLE = STANDBY
                        REPLAY_TYPE = PHYSICAL
```

```
                         HADR_SYNCMODE = SUPERASYNC
                            STANDBY_ID = 0
                         LOG_STREAM_ID = 0
                            HADR_STATE = REMOTE_CATCHUP
                   PRIMARY_MEMBER_HOST = 197.3.137.243
                      PRIMARY_INSTANCE = db210
                        PRIMARY_MEMBER = 0
                   STANDBY_MEMBER_HOST = 197.3.137.243
                      STANDBY INSTANCE = db210s
                        STANDBY MEMBER = 0
                   HADR CONNECT STATUS = CONNECTED
              HADR CONNECT STATUS TIME = 09/14/2012 18:19:51.397821(1347617991)
           HEARTBEAT INTERVAL(seconds)= 30
                 HADR TIMEOUT(seconds)= 120
         TIME SINCE LAST RECV(seconds)= 3
              PEER WAIT LIMIT(seconds)= 0
            LOG HADR WAIT CUR(seconds)= 0.000
     LOG HADR WAIT RECENT AVG(seconds)= 0.000000
    LOG HADR WAIT ACCUMULATED(seconds)= 0.000
                   LOG HADR WAIT COUNT = 0
SOCK SEND BUF REQUESTED,ACTUAL(bytes)= 0, 134752
SOCK RECV BUF REQUESTED,ACTUAL(bytes)= 0, 134752
             PRIMARY LOG FILE,PAGE,POS = S0000002.LOG, 0, 49107769
             STANDBY LOG FILE,PAGE,POS = S0000002.LOG, 0, 49107769
                   HADR LOG GAP(bytes)= 0
      STANDBY REPLAY LOG FILE,PAGE,POS = S0000002.LOG, 0, 49107769
        STANDBY RECV REPLAY GAP(bytes)= 0
                      PRIMARY LOG TIME = 09/14/2012 18:20:22.000000(1347618022)
                      STANDBY LOG TIME = 09/14/2012 18:20:22.000000(1347618022)
               STANDBY REPLAY LOG TIME = 09/14/2012 18:20:22.000000(1347618022)
          STANDBY RECV BUF SIZE(pages)= 4300
              STANDBY RECV BUF PERCENT = 0
            STANDBY SPOOL LIMIT(pages)= 0
                  PEER_WINDOW(seconds)= 0
              READS_ON_STANDBY_ENABLED = N
```

由于在 SUPERASYNC 模式下，虽然可以使用 graceful takeover 进行切换，但是在主机上提交的事务有可能无法传到备机，造成备机数据的丢失，因此这种模式一定要谨慎使用。

```
$db2 takeover hadr on db test
DB20000I  The TAKEOVER HADR ON DATABASE command completed successfully.
```

13.4.3 假脱机日志

在 DB2 V9.7 里，备机接收到的日志首先存放到备机内存的日志接收缓冲区中，日志接收缓冲区的大小由 DB2_HADR_BUF_SIZE 决定。如果日志接收缓冲区满，在某些同步模式下就会影响主机的性能。因此，DB2 V10.1 里新增加了 Log Spooling 功能，当备机日志接收缓冲区满的时候，备机会将新接收到的日志数据写到磁盘，备机重做的时候可以从磁盘读取日志信息，这样虽然对备机的重做速度有所影响，但是不会影响主机性能。

DB2 V10.1 在数据库配置参数里新增加了 hadr_spool_limit 参数来控制 Log Spooling，此参数确定允许假脱机至 HADR 备用数据库上的磁盘的最大日志数据量。

```
$db2 get db cfg for test | grep -i HADR_SPOOL_LIMIT
 HADR spool log data limit(4KB)    (HADR_SPOOL_LIMIT)= 0

$db2 update db cfg for test using HADR_SPOOL_LIMIT -1
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
SQL1363W  Database must be deactivated and reactivated before the changes to
one or more of the configuration parameters will be effective.

$db2 get db cfg for test | grep -i HADR_SPOOL_LIMIT
 HADR spool log data limit(4KB)    (HADR_SPOOL_LIMIT)= -1
```

默认值 0 意味着没有假脱机。如果缓冲区变满，那么主机上的新事务可能被阻止，因为主机不能再向备机发送任何日志数据。

值－1 意味着无限假脱机(可用磁盘空间支持的任意量)。如果对 hadr_spool_limit 使用较高值，那么应考虑主机的日志位置与备机的日志之间存在较大间隔的情况，此时可能导致接管时间较长(因为在假脱机日志的重做完成之前，备机不能充当新主机的角色)。

请注意，使用日志假脱机不会损害 HADR 功能提供的 HA/DR 保护。通过指定同步方式，主机中的数据仍以日志形式复制到备机；但是可能要多花一点时间(通过日志重演)将数据应用至表空间。

13.4.4 重做延迟

在 DB2 V9.7 中，HADR 可以很好地保护由于物理原因造成的主机异常，但是对于逻辑原因造成的数据异常，HADR 一直缺少很好的保护措施。比如 DBA 不小心删除了主机上一张表里的重要数据，同时备机由于实时同步日志，备机上的这些数据也没有了，因此只能拿头天的数据库备份进行恢复后，通过前滚日志来解决。在整个解决过程中，服务要完全停止。因此为了解决这种问题，DB2 V10.1 里新增加了重做延迟这项功能，

重做延时功能是通过新的数据库配置参数 hadr_replay_delay 来控制的，此参数指定从

数据在主数据库上更改后到这些更改反映在备用数据库上之前经历的时间(以秒数计)。

hadr_replay_delay 配置参数对 HADR 备用数据库启用延迟重演，这意味着备用数据库有意拖延 HADR 主数据库，因为备用数据库在重演主数据库中的日志。备用数据库需要处于 Super Async 方式，才能设置此参数(给予非零值)。不能对主数据库设置此参数。而且，需要对备用数据库禁用延迟重演，重新激活数据库后备用数据库才能作为主数据库接管。

如果通过设置 hadr_replay_delay 启用延迟重演，那么还应通过设置 hadr_spool_limit 数据库配置参数来启用日志假脱机。日志假脱机允许将更多日志数据传输至延迟备用数据库，所以，如果发生错误，那么可在恢复期望数据后将数据库前滚至更接近该实际错误的时间并且减少丢失的工作内容。如果将 hadr_replay_delay 设置为很大的值，那么考虑将 hadr_spool_limit 设置为－1(意味着不受限制)并提供足够的磁盘空间以在所配置延迟值的时间段保存主数据库生成的日志数据。

可以通过修改 db cfg 来修改此参数：

```
$db2 update db cfg for test using HADR REPLAY DELAY 120
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
SQL1363W  Database must be deactivated and reactivated before the changes to
one or more of the configuration parameters will be effective.

$db2 deactivate db test
DB20000I  The DEACTIVATE DATABASE command completed successfully.

$db2 activate db test
DB20000I  The ACTIVATE DATABASE command completed successfully.

$db2pd -d test -hadr | grep -i STANDBY_REPLAY_DELAY
      STANDBY_REPLAY_DELAY(seconds)= 120
```

切换的时候需要首先修改 HADR_REPLAY_DELAY 为 0，然后重启备机的 HADR，执行 takeover 才能切换。但是如果要放弃最新的数据，那么只能在备机进行前滚操作。示例如下：

```
$db2 update db cfg for test using HADR_REPLAY_DELAY 0
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
SQL1363W  Database must be deactivated and reactivated before the changes to
one or more of the configuration parameters will be effective.

$db2 deactivate db test
DB20000I  The DEACTIVATE DATABASE command completed successfully.

$db2 stop hadr on db test
```

```
DB20000I  The STOP HADR ON DATABASE command completed successfully.

$db2 deactivate db test
DB20000I  The DEACTIVATE DATABASE command completed successfully.

$db2 takeover hadr on db test
DB20000I  The TAKEOVER HADR ON DATABASE command completed successfully.
```

13.4.5 多备机

鉴于金融行业监管压力和对自身业务的保护要求，一般大型金融机构都要求建成实现两地三中心的灾备模式，即同城和异地分别需要有一个灾备中心。在 DB2 V10.1 版本前，由于源生 HADR 只能实现一主一备的模式，因此要实现两地三中心的模式，必须借助于其他工具或手段，比如 HADR+Q-Replication 或 HADR+存储级备份，这极大提高了系统复杂度和维护成本。因此，DB2 V10.1 在 HADR 功能里增加了源生的多备机支持。在使用 HADR 多备机功能后，两地三中心的灾备架构如图 13-8 所示。

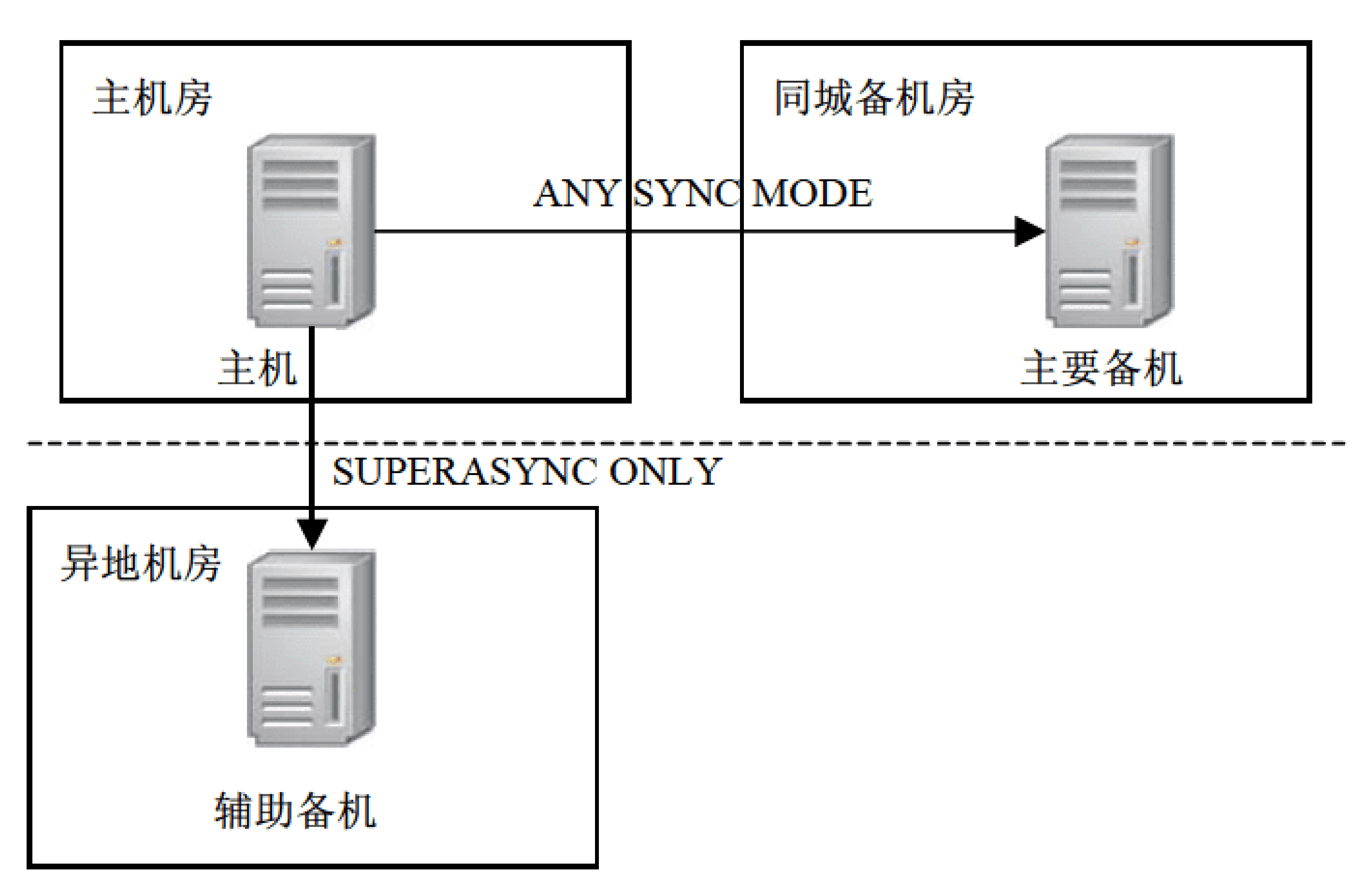


图 13-8　多备机支持架构图

在 HADR 多备机这一特性中，存在主机、主要备机和辅助备机三个角色。其中，主机真正进行业务处理；主要备机可以采用任意模式与主机进行同步，是主备切换的首选，并且可以集成到 TSA 里实现自动切换；辅助备机只能采用 SUPER ASYNC 模式进行同步，每个主机最多有一个主要备机和两个辅助备机。

主要备机的配置和 DB2 V10.1 版本之前的配置一样，主要配置 HADR_REMOTE_HOST、HADR_REMOTE_SVC 和 HADR_REMOTE_INST 这三个参数，这里不再做过多

介绍。

辅助备机的备机则是通过DB2 V10.1 里新增加的参数 HADR_TARGET_LIST 来进行配置，HADR_TARGET_LIST 使用主机名：端口号的方式进行配置，不同辅助备机之间用|分隔，可以动态配置，不需要重启实例。配置 HADR_TARGET_LIST 的语句如下：

```
UPDATE DATABASE CONFIGURATION FOR dbname USING HADR TARGET LIST
host1:port1|host2:port2
```

配置完该参数后，可以在任意辅助备机进行 graceful 或 force 切换，选用何种方式进行切换完全取决于主机当前的状态和对数据的完整性要求：

```
TAKEOVER HADR ON DB dbname
TAKEOVER HADR ON DB dbname BY FORCE
```

13.4.6　监控指标

DB2 V10.1 里对 HADR 的监控做了极大调整，增加了许多新的监控指标，表 13-9 列出了一些新增监控指标的含义。

表 13-9　HADR 监控指标

指 标 名 称	指 标 含 义
STANDBY_ID	备机的编号，在多备机环境下，每台备机对于主机来说都有一个编号，这个编号从 1 开始
HADR_CONNECT_STATUS	HADR 当前的连接状态，有 CONGESTED、CONNECTED、DISCONNECTED 三个值
HADR_CONNECT_STATUS_TIME	HADR 当前状态的开始时间
HEARTBEAT_INTERVAL	HADR 心跳时间
TIME_SINCE_LAST_RECV	在网络上，上一次收到数据的时间
LOG_HADR_WAIT_CUR	当前事务在 HADR 传输日志中等待的时间
LOG_HADR_WAIT_RECENT_AVG	最近几次等待时间的平均值
LOG_HADR_WAIT_ACCUMULATED	从 HADR 启动开始到当前时间，HADR 传输日志总共用的时间
LOG_HADR_WAITS_COUNT	一共发生了多少次等待
PRIMARY_LOG_FILE,PAGE,POS	主机上最新日志的位置
STANDBY_LOG_FILE, PAGE, POS	备机上接收的最新日志的位置
PRIMARY_LOG_TIME	主机最后事务的提交时间
STANDBY_LOG_TIME	备机最后事务的回放时间
STANDBY_RECV_BUF_SIZE	备机日志缓冲区的大小

(续表)

指 标 名 称	指 标 含 义
STANDBY_SPOOL_LIMIT	Spooling 的大小，0 代表没有 Spooling
READS_ON_STANDBY_ENABLED	备机是否可读
STANDBY_REPLAY_DELAY	备机重做延迟时间

13.5 凌波微步——性能增强

“凌波微步者，迅如疾风，速如闪电，眨眼之间可达属地”。DB2 V10.1 中的性能增强部分就是 DB2 的“凌波微步”，能让 DB2 运行得更快、更好、更强。DB2 V10.1 包含众多 SQL 性能增强功能，这使 DB2 数据服务器继续作为适合于任意规模组织机构的业界领先的数据服务器解决方案。DB2 SQL 查询优化器已进行改进，现在提供了下列增强功能：

13.5.1 提高了一组常用 SQL 语句的查询性能

在 DB2 V10.1 中，包括了众多性能改进以提高许多查询的速度。

现在，使用高效率的散列函数在查询处理的早期阶段除去部分重复项。这可能不会除去全部重复项，但会减少以后在查询求值阶段必须处理的数据量。除去某些初始重复行可以提高查询速度并降低耗尽排序堆内存的机会，从而在这类情况下不需要使用速度相对较低的磁盘空间作为临时存储器。此项改进称为“早期非全面消除重复”(PED)。

要确定是否正在将此项改进用于特定查询，请激活 Explain 设施并运行该查询。将此项新功能应用于查询后，EXPLAIN_ARGUMENT 表中的如下新值将指示此情况：

ARGUMENT_TYPE 列= UNIQUE

ARGUMENT_VALUE 列现在还可以包含 HASHED PARTIAL 值，这表示已利用新功能。

现在，db2exfmt 工具将在其输出中显示 HASHED PARTIAL，如以下示例所示：

```
7)UNIQUE:(Unique)
    Cumulative Total Cost:   175.634
Cumulative CPU Cost:   1.95446e+06
    ...
    ...
      Arguments:
      ---------
      JN INPUT:(Join input leg)
           INNER
```

```
        UNIQKEY :(Unique Key columns)
             1: Q1.C22
        UNIQKEY :(Unique Key columns)
             2: Q1.C21
        UNIQUE  :(Uniqueness required flag)
   HASHED PARTIAL
```

这些改进是自动进行的；不需要配置任何设置，也不需要更改 SQL 语句。

13.5.2　RUNSTATS 支持索引采样

现在，RUNSTATS 命令可以使用采样方法(而不必扫描整个索引)来收集索引统计信息。可以使用新命令参数 INDEXSAMPLE 来激活此功能，此接口与现有的 TABLESAMPLE 命令参数类似。通常，新的采样方法能够减少 RUNSTATS 处理的叶子节点总数(假设指定了 INDEXSAMPLESYSTEM)或索引条目总数(假设指定了 INDEXSAMPLEBERNOULLI)，从而缩短生成统计信息所需的时间。

从 DB2 V10.1 开始，用于收集详细索引统计信息的默认方法已更改。使用 DETAILED 选项时，将不再扫描整个索引，而是使用采样方法来收集统计信息。现在，此选项相当于为了实现兼容性而保留的 SAMPLEDDETAILED 选项。如果要像先前发行版一样通过扫描整个索引来收集详细的索引统计信息，可以指定 UNSAMPLED 选项。

现在，RUNSTATS 命令支持 VIEW 命令参数。添加此参数的意图是，使你对视图运行 RUNSTATS 时更为直观。此命令的运行方式就像是为视图指定了 TABLE 参数一样。

为改进 RUNSTATS 命令的可用性，不必再通过指定模式名来完全限定对象名。如果未指定模式名，将使用默认模式。

13.5.3　优化概要文件能支持注册变量和非精确匹配

现在，优化概要文件可用于设置某些注册变量并支持非精确匹配。非精确匹配可用于在编译查询语句时更好地进行匹配。

可以使用 REGISTRY 元素中的 OPTION 元素在优化概要文件中设置部分注册变量。OPTION 元素具有 NAME 和 VALUE 属性，可以在这些属性中指定注册变量及其值。可以在全局级别设置许多注册变量，另外，对于特定的语句，也可以在语句级别设置这些变量。

现在，优化概要文件除了支持精确匹配以外，还支持非精确匹配。非精确匹配将在匹配语句时忽略文字、主变量和参数标记。要在优化概要文件中指定非精确匹配，请将 STMTMATCH 元素的 EXACT 属性值设置为 FALSE。可以在全局级别和语句级别同时指定 STMTMATCH 元素。

13.5.4 统计视图改进了统计信息以及查询优化器的统计信息收集

统计视图提供了新的功能部件，DB2 查询优化器现在可以使用这些功能部件来生成更好的访问方案，从而提高某些查询的性能。

- 包含复杂表达式的谓词：现在，DB2 查询优化器可以使用统计视图中的表达式列(带有一个或多个函数的列)中的统计信息。在先前的发行版中，对于谓词包含复杂表达式的查询，优化器只能使用默认值进行选择性估算。但是，从此发行版开始，优化器现在可以使用实际统计信息来生成更好的访问方案。
- 减少统计视图的数目：现在，如果存在引用完整性约束并且在数据中定义了这些约束，那么可以减少获取星型连接查询的良好统计信息所需的统计视图数目。现在，可以创建统计视图并在其中包含许多来自连接查询的列。特定连接的统计信息将根据引用完整性约束从这个统计视图推断。
- 对统计视图收集的列组统计信息：现在，DB2 查询优化器可以使用对视图收集了列组统计信息的统计视图中的统计信息。将列组统计信息与统计视图进行组合可以改进访问方案，这是因为优化器可以使用从可能有所偏差的查询收集到的调整后统计信息。
- 统计视图的自动统计信息收集：现在，DB2 自动统计信息收集功能可以自动收集统计视图的统计信息。在默认情况下，此功能未处于启用状态，而必须使用新的数据库配置参数 auto_stats_view 开启。必须使用 UPDATE 命令来开启这个新参数，这样才能从统计视图中自动收集统计信息。自动统计信息收集功能收集的统计信息相当于发出以下命令：runstats on view <*view_name*> with distribution。

13.5.5 分区内并行性改进

DB2 查询优化器的目标之一是，选择并行执行策略以维护子代理程序之间的数据平衡并保持它们的繁忙程度相同。在此发行版中，对优化器的并行化能力作了进一步增强，以使更多工作负载能够更充分地利用多核心处理器。

- 对不平衡的子代理程序的工作负载进行重新平衡：数据过滤和数据偏差可能会导致子代理程序之间的工作负载在查询执行期间变得不平衡。不平衡的工作负载的效率欠佳，而连接和其他计算成本较高的操作会使这种情况加重。优化器将在查询的访问方案中查找不平衡的根源并应用平衡策略，从而确保在子代理程序之间均匀地分配工作。对于无序的外部数据流而言，优化器通过对外部数据流使用 REBAL 运算符来平衡连接。对于有序数据流(有序数据由索引访问或排序生成)而言，优化器将使用共享排序来平衡数据。如果排序溢出到临时表中，那么将不会使用共享排序，这是因为排序溢出的成本较高。

- 对范围分区表和索引执行的并行扫描：可以对范围分区表运行并行表扫描，同样，可以对分区索引运行并行索引扫描。对于并行扫描而言，分区索引将根据索引键值以及键值的键条目数划分为多个记录范围。并行扫描开始时，将记录范围分配给子代理程序，子代理程序完成处理某个范围时，将被赋予新范围。索引分区按顺序进行扫描，并且子代理程序可能在任意时间点扫描未保留的索引分区，而不会相互等待。只有与查询相关的索引分区子集(由数据分区消除分析确定)才会被扫描。
- 能够对并行度进行调节以便针对事务性工作负载进行优化：现在，各个应用程序或工作负载可以动态地调节分区内并行度，以便针对正在执行的查询的类型来优化性能。在先前版本的 DB2 中，只能针对整个实例控制并行度(以及是将其开启还是关闭)。开启或关闭并行性也要求重新启动实例。在具有混合工作负载的数据库服务器上，需要更灵活的方法来控制分区内并行性。事务性工作负载(通常包括较短的插入、更新和删除事务)无法从并行性中受益。启用分区内并行性后，存在一些处理开销，这将对事务性工作负载产生负面影响。但是，由于数据仓库工作负载通常包括长时间运行的处理器密集型查询，因此将由于并行化而大大受益。对于包含事务性成分和数据仓储成分的混合工作负载而言，现在可以对数据库系统进行配置，以便提供对于每个应用程序部署的工作负载类型而言最优的并行性设置。可以通过应用程序逻辑来控制并行性设置，也可以通过 DB2 工作负载管理器进行此控制(这不要求更改应用程序)。
- 从数据库应用程序中控制分区内并行性：要从数据库应用程序中启用或禁用分区内并行性，可以调用新的 ADMIN_SET_INTRA_PARALLEL 存储过程。例如，以下语句将启用分区内并行性：

  ```
  CALL ADMIN_SET_INTRA_PARALLEL('YES')
  ```

  虽然在当前事务中调用此存储过程，但此存储过程从下一个事务开始生效，并且仅适用于主调应用程序。ADMIN_SET_INTRA_PARALLEL 进行的分区内并行性设置将覆盖 intra_parallel 配置参数中的值。
- 从 DB2 工作负载管理器中控制分区内并行性：要对指定的工作负载启用或禁用分区内并行性，可以设置 MAXIMUM DEGREE 工作负载属性。例如，以下语句对名为 trans 的工作负载禁用分区内并行性：

  ```
  ALTER WORKLOAD trans MAXIMUM DEGREE 1
  ```

  在 ALTER WORKLOAD 语句之后执行的工作负载中的所有语句都将在分区内并行性处于关闭状态的情况下运行。MAXIMUM DEGREE 工作负载属性进行的分区

内并行性设置将覆盖对 ADMIN_SET_INTRA_PARALLEL 进行的调用，并且将覆盖 intra_parallel 配置参数中的值。

13.5.6 通过更有效地进行数据和索引预取来提高查询性能

DB2 V10.1 提供了敏捷数据预取和敏捷索引预取功能，这提高了查询性能，并且降低了重组表和索引的需要。

在对表数据或索引进行许多更改之后，连续数据或索引有可能位于集群状况不佳的数据页或低密度索引叶子页上。在先前的发行版中，这可能会导致查询性能下降。这是因为随着集群状况不佳的数据页增加以及索引叶子页的密度降低，顺序检测预取的效率将较低。

仅在 ISCAN-FETCH 期间应用敏捷数据预取，但在任何索引扫描(即使是 ISCAN-FETCH 的一部分)期间都将引用敏捷索引预取。优化器可以将敏捷数据预取和敏捷索引预取相结合，以便选择最佳的索引预取和数据预取技术。但是，敏捷索引预取和敏捷数据预取互不相关。

DB2 V10.1 中引入了一种称为提前读预取的新型预取，可用来高效预取集群状况不佳的数据页和低密度的索引页。除了下面说明的限制之外，优化器将选择提前读预取作为顺序检测预取的备用方法。在运行时，如果检测到顺序检测预取的运行情况不是足够良好，那么预取类型可能会从顺序检测预取切换为提前读预取。提前读预取将预先了解索引以确定索引扫描操作将访问的准确数据页或索引叶子页，并且预取这些数据页和索引叶子页。虽然提前读预取提供了在执行索引扫描期间需要的所有数据页和索引叶子页(没有不需要的页)，但是还需要其他资源才能找到这些页。对于高度连续的数据或索引，顺序检测预取的性能通常将超过提前读预取的性能。

采用敏捷数据预取方法时，将根据数据集群程度来确定是使用顺序检测预取还是提前读预取。当连续存储数据页时，将使用顺序检测预取；当数据页的集群不佳时，将使用提前读预取。敏捷数据预取使数据库系统能够充分利用存储在连续页中的数据的潜在性能优势，同时又能高效预取集群状况不佳的数据。由于集群状况不佳的数据不再对查询性能不利，因此降低了执行成本高昂的操作(例如对表进行重组)的需要。

采用敏捷索引预取方法时，将根据索引的密度来确定是使用顺序检测预取还是提前读预取。当连续存储索引时，将使用顺序检测预取；当索引的密度降低时，将使用提前读预取。敏捷索引预取使数据库系统能够充分利用连续存储的索引的潜在性能优势，同时又能高效预取低密度的索引。敏捷索引预取方法降低了执行成本高昂的操作(例如对索引进行重组)的需要。

敏捷数据预取和敏捷索引预取仅适用于索引扫描操作，而不支持 XML 索引、扩展索引和文本搜索文本索引。对全局范围集群表索引进行扫描期间，由于这些索引是逻辑索引，

而不是物理索引，因此无法使用敏捷数据预取。此外，对于敏捷数据预取，如果 ISCAN-FETCH 对全局范围分区索引进行扫描，那么将不会使用数据提前读预取。如果在敏捷数据预取的索引扫描期间对索引谓词进行了求值，并且优化器确定并不是太多行符合该索引扫描的要求，那么将禁用提前读预取。敏捷索引预取也无法用于范围集群表索引。

13.5.7　提高了对具有组合索引的表执行的查询的性能

DB2 查询优化器现在可创建其他存取方案，通过使用跳跃扫描操作，这些存取方案对开始键与结束键之间存在索引间隔的查询可能更高效。例如，在针对具有组合索引的表发出的包含多个谓词的查询中，索引间隔很常见。跳跃扫描使得不需要索引间隔规避措施，例如创建其他索引。

1. 问题：索引间隔

对于涉及许多即席查询的工作负载而言，通常很难对数据库进行优化以获得较高性能。对具有组合(多列)索引的表执行的查询就是一项特殊的挑战。理想情况下，查询的谓词与表的组合索引一致。这意味着每个谓词都可以用作开始-结束键，这反过来将缩小需要搜索的索引范围。当查询包含与组合索引不一致的谓词时，这就称为*索引间隔*。就其本身而论，索引间隔是查询特征，而不是表索引特征。

例如，假定存在表 T，其中包含整数列 A、B 和 C，并且对列 A、B 和 C 定义了组合索引。现在，请考虑以下针对表 T 执行的查询：

```
SELECT * FROM t WHERE a=5 AND c=10
```

此查询在组合索引的列 B 处有索引间隔(这假定访问方案包含对组合索引执行索引扫描)。

如果存在索引间隔，那么索引扫描可能必须处理许多不必要的键。可能需要对索引中每个符合开始/结束键条件的键单独应用索引的非前导列中的谓词。这将降低索引扫描速度，因为需要处理更多的行，并且需要为每个键对其他谓词进行求值。另外，DB2 还必须按顺序检查某个较大范围内的所有键。

要避免索引间隔，可以定义其他索引以涵盖工作负载中可能出现的查询谓词的排列。这并非理想解决方案，因为定义其他索引将产生数据库管理开销并且会消耗存储器容量。另外，对于涉及许多即席查询的工作负载而言，可能难以预计需要的索引。

2. 解决方案：启用跳跃扫描

在 DB2 V10.1 中，查询优化器可构建存取方案，该存取方案在查询包含索引间隔时使

用跳跃扫描操作。在跳跃扫描操作中，索引管理器将为包含索引间隔的小部分组合索引标识合格的键，然后使用这些合格的键填充这些间隔。最终结果是，索引管理器将跳过不会生成任何结果的索引部分。

注意：

对查询进行求值时，可能会存在查询优化器构建不包含跳跃扫描操作的访问方案的情况，即使存在索引间隔也是如此。如果查询优化器认为另一种方法比使用跳跃扫描效率更高，那么将会出现这种情况。

13.5.8 提高了基于星型模式的查询的性能

经过改进的星型模式检测算法允许查询优化器检测基于星型模式的查询并利用特定于星型模式的策略，从而提高这些查询的性能。另外，为了在数据仓库和数据集市环境中提高那些使用了星型模式的查询的性能，可以使用新的 Zigzag 连接方法将一个或多个事实表与两个或两个以上的维度表相连接。

经过改进的星型模式检测

经过改进的星型模式检测算法并不是使分析根据表大小来确定查询是否基于星型模式，而是依赖于维/雪花表的主键、唯一索引或唯一约束以及维/雪花表与事实表之间的连接谓词。经过增强的星型检测算法能够识别查询块中的多个星型，去除了 DB2 Database for Linux, UNIX, and Windows V10.1 之前使用的星型检测算法所实施的某些限制。如果新的检测方法无法检测查询是否基于星型模式，例如，如果维表上不存在主键、唯一索引或唯一约束，那么将改为使用旧检测方法。通过使用跳跃扫描功能，查询优化器能够识别星型模式，即使查询中缺少连接谓词。

新的 Zigzag 连接方法

在此发行版的 DB2 Database for Linux, UNIX, and Windows 之前，使用两种特定的策略来处理星型模式连接查询：

- 笛卡尔中心连接计划：此计划计算各个维的笛卡尔乘积，然后使用笛卡尔乘积中的每一行来探测多列事实表索引。
- 星型连接计划：此计划按维对事实表进行预先过滤以生成半连接，然后在索引与这些半连接的结果之间执行 AND 运算，最后完成这些半连接。

除了这两种特殊的星型连接处理技术以外，现在可以使用新的 Zigzag 连接方法加快基于星型模式的查询的处理速度。

Zigzag 连接是一种连接方法，其中以星型模式连接了一个事实表及两个或更多个维表，以便使用索引访问事实表。Zigzag 连接要求在每个维表与事实表之间使用等式谓词，此连接方法在不将笛卡尔乘积实际具体化的情况下计算维表中各行的笛卡尔乘积，并使用多列索引来探测事实表，以便同时按两个或两个以上的维表过滤事实表。事实表的探测器将找到匹配的行。然后，Zigzag 连接返回事实表索引中的下一个值组合。这个值组合称为反馈，用于跳过维表笛卡尔乘积提供的那些在事实表中找不到匹配项的探测器值。同时按两个或两个以上维表对事实表进行过滤，以及跳过已知没有效益的探测器，共同使 Zigzag 连接成为查询大型事实表的高效方法。在默认情况下，Zigzag 连接处于启用状态。

13.6　火眼金睛——监控增强

话说当前孙悟空保护唐僧西天取经，一路困难重重，全是依靠自己的火眼金睛，一路上识别了多少妖魔鬼怪，才最终保护唐僧顺利到达西天求经。在数据库运维中，监控便是“西天求经”的“火眼金睛”，我们需要通过运维来保障数据库日常运行的稳定，规避一些潜在的风险。

DB2 V10.1 包括许多增强功能，这些功能借助更详细的控制来更全面地监视 DB2 数据库环境。下列监视增强功能提供了新的监视信息：

- 用于跟踪配置更改的事件监视器
- 用法列表对象记录影响表或索引的语句
- 使用新的 STATEMENT 阈值域为特定语句创建阈值
- 用于访问监视信息的新函数和已更改的函数
- 工作单元事件监视器捕获的信息中现在包括的可执行标识列表
- 所有事件监视器的写至表支持
- 使用 ALTER EVENT 监视器语句修改事件监视器捕获的信息作用域的能力
- 对先前发行版中创建的事件监视器输出表进行升级
- 修改未格式化的事件表中的数据
- 用于使你更详细地了解 DB2 服务器的新监视元素

13.6.1　用于跟踪配置更改的事件监视器

在日常工作中，有时候，尤其是测试数据库时，往往很多人都有权限去访问，这就存在一定的风险问题，有可能很多人不经过同意就对数据库进行了某些变更。在之前的版本中，数据库无法记录这些配置的变更。在 DB2 V10 中，新的变更历史记录事件监视器能捕获可能影响常规数据库工作负载运行的更改。可使用此事件监视器来捕获与数据库和数据

库管理系统的行为、性能或稳定性更改一致的事件的相关数据。

常规工作负载遇到性能降低或发现意外行为时，需要确定发生了哪些可能导致该问题的更改。以下更改可能对数据库系统有负面影响：

- 意外创建或删除索引
- 安排维护运行失败
- 更改数据库配置参数或 DB2 注册变量

更改可能由用户显式导致。例如，管理员可能运行删除索引的 DDL，或者更改可能在没有任何用户交互的情况下隐式发生或自动发生。例如，自调整内存管理器(STMM)可能会更改配置参数，自动表重组可能会重组表。

尽管并非故意，但任何未预期更改可能会难以手动跟踪。例如，配置更新会写至诊断日志文件(db2diag.log)，实用程序进度在数据库历史记录文件中捕获。变更历史记录事件监视器提供单个界面来捕获更改数据库系统的行为和性能特征的事件。通过使用事件监视器表，可查看自己关心的任何更改事件。

变更历史记录事件监视器可捕获许多操作的与更改相关的事件，包括：

- 数据库和数据库管理器配置参数更改
- 注册变量更改
- DDL 语句的执行
- 变更历史记录事件监视器的启动
- 执行以下 DB2 实用程序和命令
  - ◊ LOAD
  - ◊ 移动表(ADMIN_MOVE_TABLE 存储过程调用)
  - ◊ 联机备份
  - ◊ 联机复原
  - ◊ 联机前滚
  - ◊ 再分发
  - ◊ REORG
  - ◊ RUNSTATS

通常，不会捕获在变更历史记录事件监视器处于不活动状态或数据库脱机时发生的事件的相关信息，但是会记录对注册变量和配置参数的更改。

如下是在 DB2 V10.1 中配置变更的监控示例：在这个示例中创建了针对数据库配置变更的监控器，监控器将记录配置的变更和初始值。

```
CREATE EVENT MONITOR CFG_WITH_OFFLINE
   FOR CHANGE HISTORY WHERE EVENT IN(CFGALL)
```

```
WRITE TO TABLE
  CHANGESUMMARY(TABLE CHG_SUMMARY_HISTORY),
  DBDBMCFG(TABLE DB_DBM_HISTORY),
  REGVAR(TABLE REGVAR_HISTORY)
AUTOSTART;
```

13.6.2　用法列表对象记录影响表或索引的语句

DB2 V10.1 使用新的用法列表数据库对象，可以记录引用的特定表或索引在 DML 语句中的特定部分，并且可以在这些语句部分执行时捕获关于它们的统计信息。

用法列表中的每个条目都包含关于该部分在特定时间段内执行次数的信息。另外，这些条目还包含统计信息总计，用于指出该部分在所有执行期间如何对表或索引产生影响。

用法列表还包含关于一些因素(例如每个语句部分的锁定和缓冲池使用情况)的统计信息。如果确定某条语句对表或索引产生负面影响，请使用这些统计信息确定可能需要进一步监视哪些地方，或是确定调整该语句的方式。

使用一些用法列表以在影响特定表的 DML 语句段执行时标识这些语句段。可查看每条语句的统计信息并使用这些统计信息来确定可能需要其他监视或调整的位置。

在开始监控之前，我们需要执行以下任务：

- 标识要查看对象用法统计信息的表。可使用 MON_GET_TABLE 表函数来查看一个或多个表的监视度量值。
- 要发出必需的语句，确保每条语句的授权标识持有的特权包括 DBADM 权限或 SQLADM 权限。
- 确保对 MON_GET_TABLE_USAGE_LIST 和 MON_GET_USAGE_LIST_ STATUS 表函数具有 EXECUTE 特权。

在查看 MON_GET_TABLE 表函数的输出时，可能会见到不寻常的监视元素值。可使用一些用法列表来确定是否有任何 DML 语句影响了此值。

用法列表包含有关特定时间段内影响表的每条语句的锁定和缓冲池使用情况之类的因子的统计信息。如果确定某条语句对表有负面影响，请使用这些统计信息来确定是否需要进一步监视或如何调整此语句。

具体的监控操作如下：

要标识影响表的语句，请执行以下操作：

- 通过发出以下命令，将 MON_OBJ_METRICS 配置参数设置为 BASE：

```
DB2 UPDATE DATABASE CONFIGURATION USING MON_OBJ_METRICS BASE
```

- 通过使用 CREATE USAGE LIST 语句来为表创建用法列表。 例如，要为 SALES. INVENTORY 表创建 INVENTORYUL 用法列表，请发出以下命令：

```
CREATE USAGE LIST INVENTORYUL FOR TABLE SALES.INVENTORY
```

- 通过使用 SET USAGE LIST STATE 语句来激活对象用法统计信息的收集。 例如，要激活针对 INVENTORYUL 用法列表的收集，请发出以下命令：

```
SET USAGE LIST INVENTORYUL STATE = ACTIVE
```

- 在收集对象统计信息期间，使用 MON_GET_USAGE_LIST_STATUS 表函数来确保用法列表处于活动状态并且已对此用法列表分配足够内存。例如，要检查 INVENTORYUL 用法列表的状态，请发出以下命令：

```
SELECT MEMBER,
       STATE,
       LIST SIZE,
       USED ENTRIES,
       WRAPPED
FROM TABLE(MON_GET_USAGE_LIST_STATUS('SALES', 'INVENTORYUL', -2))
```

- 经历要收集对象用法统计信息的时间段后，应使用 SET USAGE LIST STATE 语句来取消激活用法列表数据的收集。例如，要取消激活对 INVENTORYUL 用法列表的收集，请发出以下命令：

```
SET USAGE LIST SALES.INVENTORYUL STATE = INACTIVE
```

- 使用 MON_GET_TABLE_USAGE_LIST 函数来查看收集的信息。可查看收集统计信息的时间段内影响表的一部分或全部语句的统计信息。 例如，如果只想查看读取最多表行的 10 条语句，请发出以下命令：

```
SELECT MEMBER,
      EXECUTABLE ID,
      NUM REFERENCES,
      NUM REF WITH METRICS,
      ROWS READ,
      ROWS INSERTED,
      ROWS UPDATED,
      ROWS DELETED
FROM TABLE(MON GET TABLE USAGE LIST('SALES', 'INVENTORYUL', -2))
ORDER BY ROWS READ DESC
FETCH FIRST 10 ROWS ONLY
```

- 如果要查看影响此表的语句的文本，请将 MON_GET_TABLE_USAGE_LIST 输出中的 executable_id 元素的值用作 MON_GET_PKG_CACHE_STMT 表函数的输入。例如，发出以下命令来查看特定语句的文本：

```
SELECT STMT_TEXT
FROM TABLE
(MON_GET_PKG_CACHE_STMT(NULL,
x'01000000000000007C0000000000000000000000020020081126171720728997', NULL,
-2))
```

- 使用语句列表和为语句提供的统计信息来确定需要额外监视或调整的位置。例如，pool_writes 监视元素的值很低(相对于 direct_writes 监视元素的值)的语句可能存在需要注意的缓冲池问题。

在监控完成之后，在不需要用法列表中的信息时，请使用 SET USAGE LIST STATE 语句释放与用法列表相关联的内存。例如，要释放与 INVENTORYUL 用法列表相关联的内存，请发出以下命令：

```
SET USAGE LIST SALES.INVENTORYUL STATE = RELEASED
```

13.6.3　使用新的 STATEMENT 阈值域为特定语句创建阈值

在数据库性能优化过程中经常需要查找那些运行时间长的语句，不管是通过 SNAPSHOT 还是其他方法，这个查找过程非常繁杂和令人厌烦。在 DB2 V10.1 中，CREATE THRESHOLD 语句语法中添加了名为 STATEMENT 的新阈值域。通过这个监控我们可以定义自己的阀值，找出那些大于这个阀值的语句，更加方便我们去优化。

例如，可对“SELECT * FROM TABLE1, TABLE2”之类的 SQL 语句定义 CPUTIME 阈值，这样一来，如果执行此语句时超过了语句的 CPU 时间阈值，那么会发生阈值违例。可通过指定语句文本或语句的可执行标识来针对这些阈值标识该语句。与其他域的阈值类似，可配置 STATEMENT 阈值以将违反该阈值的活动的相关信息写至活动事件监视器。

与较低发行版相比，这个新功能在捕获信息方面更详细且更具体。在先前的发行版中，确定特定语句的活动存在的问题需要捕获许多活动的相关信息，然后仔细检查事件监视器数据以查找异常情况。现在，标识运行时长超过预期的语句后，可迅速收集并检查仅与该语句相关的活动信息。例如，可以查看在语句中指示产品标识的参数标记所表示的数据。或者，你可能会发现检查与语句执行相关的“耗用时间”监视元素非常有用，例如总执行时间(TOTAL_EXEC_TIME)。

如果发现某条语句的执行时间很长，那么可定义阈值以在超出该阈值时让活动事件监视器捕获有关该语句的执行信息。然后，可将语句执行信息与活动事件监视器收集的信息

相关联以帮助了解可能导致速度变慢的原因。

在监控之前，需要确定哪些 SQL 语句导致性能降低，这可能是从业务人员中获得，也可能是从数据库监控中抓取运行时间长的语句.

在下面的示例中，正被调查的查询作为应用程序的一部分运行。查询如下所示：

```
SELECT DISTINCT PARTS_BIN FROM STOCK WHERE PART_NUMBER = ?
```

速度变慢的可能原因之一是数据分发不适宜。例如，如果 STOCK 表只有几行对应大部分部件编号，但有几千行对应某个特定部件编号，那么需要花较长时间来运行此 SELECT 语句。以下示例说明如何检索与先前查询相关联的活动针对参数标记(“?”)正在处理的实际值。

要测试不适宜数据分发导致查询运行变慢的假设，可为提到的语句创建阈值。然后，可使用阈值和活动事件监视器来捕获有关该特定语句的执行信息。可通过此信息确定运行时间超过预期的查询处理的实际值。

- 为提到的语句创建阈值，指定语句运行超过 10 秒时应发生的阈值违例事件：

```
CREATE THRESHOLD TH1
    FOR STATEMENT TEXT 'SELECT DISTINCT PARTS BIN
    FROM STOCK WHERE PART NUMBER = ?' ACTIVITIES
    ENFORCEMENT DATABASE
    WHEN ACTIVITYTOTALTIME > 10 SECONDS
    COLLECT ACTIVITY DATA WITH DETAILS, SECTION AND VALUES
    CONTINUE
```

- 创建阈值事件监视器来记录阈值违例：

```
CREATE EVENT MONITOR STMT_THRESH_VIOLATIONS
FOR THRESHOLD VIOLATIONS
WRITE TO TABLE
AUTOSTART
```

- 创建活动事件监视器来记录详细活动信息：

```
CREATE EVENT MONITOR ACTIVITIES
FOR ACTIVITIES
WRITE TO TABLE
```

- 启用新事件监视器：

```
SET EVENT MONITOR ACTIVITIES STATE 1
SET EVENT MONITOR STMT_THRESH_VIOLATIONS STATE 1
```

- 运行执行此语句的应用程序。如果发生阈值违例，那么阈值违例事件监视器 STMT_THRESH_VIOLATIONS 会捕获有关阈值违例的信息，活动事件监视器 ACTIVITIES 会记录与阈值违例相关联的活动的信息。
- 要确定是否发生了阈值违例，请查询前面阈值事件监视器记录的阈值 TH1 的违例次数。为此，必须将视图 SYSCAT.THRESHOLDS 与阈值事件监视器生成的包含阈值违例信息的表连接到一起，因为阈值名称 TH1 保留在 SYSCAT.THRESHOLDS 中：

```
SELECT COUNT(1)NUM VIOLATIONS
      FROM THRESHOLDVIOLATIONS DB2THRESHOLDVIOLATIONS T
      JOIN SYSCAT.THRESHOLDS S ON T.THRESHOLDID = S.THRESHOLDID
      WHERE S.THRESHOLDNAME = 'TH1';
NUM VIOLATIONS
--------------
                  1

  1 record(s)selected.
```

在此情况下，有阈值违例，发现一次语句执行的运行时间超过 10 秒。

- 检查语句内参数标记(?)表示的数据(部件号)。在以下示例中，SELECT 语句从活动事件监视器生成的其中一个 ACTIVITYVALS 表中检索参数标记(由如下 SQL 中的 STMT_VALUE_DATA 表示)的值：

```
SELECT SUBSTR(V.STMT VALUE DATA, 1, 80)PARAM MARKER VALUE
      FROM ACTIVITYVALS ACTIVITIES V
      JOIN THRESHOLDVIOLATIONS STMT THRESH VIOLATIONS T
        ON T.APPL ID = V.APPL ID
        AND T.UOW ID = V.UOW ID
        AND T.ACTIVITY ID = V.ACTIVITY ID
      JOIN SYSCAT.THRESHOLDS S
        ON T.THRESHOLDID = S.THRESHOLDID
      WHERE S.THRESHOLDNAME = 'TH1';
```

在先前示例中，SELECT 语句从活动事件监视器生成的某个表中检索参数标记(STMT_VALUE_DATA)的值：

```
PARAM MARKER VALUE
-----------------------------------------------------------------------
475299
```

- 既然知道与长时间运行的语句相关联的 PART_NUMBER 的值，那么可检查 STOCK 表以了解表中是否存在任何与该部件编号出现次数有关的因素，这些因素可能导

致查询时间变长。例如，包含 475299 作为 PART_NUBMER 的值的大量行(与其他部件编号的行数相比)可能成为查询运行时间变长的原因(如果遇到此值)。

13.6.4 用于访问监视信息的新函数和已更改的函数

表 13-10 描述了 DB2 V10.1 中引入的新表函数，它们返回其他监视信息。

表 13-10 用于访问监视信息的新表函数

名 称	详 细 信 息
ADMIN_GET_STORAGE_PATHS	此表函数返回每个数据库存储器组的自动存储器路径列表，包括每个存储器路径的文件系统信息
MON_GET_AUTO_MAINT_QUEUE	此表函数返回当前已排队以便由自动计算守护程序(db2acd)执行的所有自动维护作业(除实时统计信息作业以外)的相关信息
MON_GET_AUTO_RUNSTATS_QUEUE	此表函数返回当前已排队以便由当前相连数据库中的自动统计信息收集功能进行评估的所有对象的相关信息
MON_GET_CF	此表函数返回关于系统中的一个或多个集群高速缓存设施的状态信息
MON_GET_CF_CMD	此表函数报告集群高速缓存设施处理某个请求所耗用的时间(以毫秒计)
MON_GET_CF_WAIT_TIME	此表函数报告等待集群高速缓存设施处理某个请求所耗用的时间(以毫秒计)以及与集群高速缓存设施进行相关通信所耗用的时间
MON_GET_EXTENDED_LATCH_WAIT	此表函数返回关于过长等待中涉及的锁存器的信息
MON_GET_GROUP_BUFFERPOOL	此表函数返回有关组缓冲池的信息
MON_GET_HADR	此表函数返回高可用性灾难恢复信息
MON_GET_INDEX_USAGE_LIST	此表函数返回对索引定义的用法列表中的信息
MON_GET_PAGE_ACCESS_INFO	此表函数返回关于指定表正在等待的缓冲池页面的信息
MON_GET_REBALANCE_STATUS	此表函数返回针对表空间的重新平衡操作的状态
MON_GET_RTS_RQST	此表函数返回关于系统中所有正处于暂挂状态的实时统计信息请求的信息， 以及关于实时统计信息守护程序当前正在处理的一组请求的信息
MON_GET_SERVERLIST	此表函数返回关于当前相连的数据库的服务器列表(高速缓存在一个或多个成员上)的度量值

(续表)

名　　称	详 细 信 息
MON_GET_TABLE_USAGE_LIST	此表函数返回对表定义的用法列表中的信息
MON_GET_TRANSACTION_LOG	此表函数返回关于当前相连的数据库的事务日志记录子系统的信息
MON_GET_USAGE_LIST_STATUS	此表函数返回关于用法列表的信息，例如列表大小、上次更改时间以及为该列表分配的内存量
MON_SAMPLE_SERVICE_CLASS_METRICS	此表函数从跨一个或多个数据库的一个或多个服务类中读取两个时间点的系统度量值，并且根据这些度量值计算各种统计信息
MON_SAMPLE_WORKLOAD_METRICS	此表函数从跨一个或多个数据库的一个或多个工作负载中读取两个时间点的系统度量值，并且根据这些度量值计算各种统计信息

13.6.5　工作单元事件监视器捕获的信息中现在包括的可执行标识列表

从 DB2 V10.1 收集工作单元内的可执行标识列表以及关联的语句级别度量值。可以使用以下两种机制的其中一种来控制这些信息的收集：

- 通过设置 mon_uow_data 和 mon_uow_execlist 数据库配置参数在数据库级别启用收集。
- 通过在 CREATE 或 ALTER WORKLOAD 语句中使用 COLLECT UNIT OF WORK DATA 子句来启用对特定工作负载的收集。

要启用对可执行标识信息的收集，请将 mon_uow_data 设置为 BASE，并将 mon_uow_execlist 设置为 ON，如以下示例所示：

```
UPDATE DB CFG FOR SAMPLE USING mon uow data BASE
UPDATE DB CFG FOR SAMPLE USING mon_uow_execlist ON
```

CREATE 和 ALTER WORKLOAD 语句中 COLLECT UNIT OF WORK DATA 子句的语法也已更改为指示收集可执行标识信息。

在分区数据库环境中，将为每个成员(协调程序或数据成员)收集可执行标识列表。在 DB2 pureCluster 环境中，系统会从协调程序成员处收集可执行标识列表。

13.6.6　使用 ALTER EVENT 监视器语句修改事件监视器捕获的信息作用域

创建写至表的事件监视器时，可指定应从事件监视器输出中排除来自一个或多个逻辑数据组的数据。从 DB2 V10.1 开始，可使用新的 ALTER EVENT MONITOR 语句以添加先

前从事件监视器中排除的逻辑数据组。在之前的发行版中，要添加先前排除的数据组，必须删除事件监视器，然后重新创建。

例如，如果创建写至表的锁定事件监视器，那么可指定仅应捕获来自 lock_participants 逻辑数据组的元素。在这种情况下，事件监视器将仅创建 LOCK_PARTICIPANTS_evmon-name 表，其中 evmon-name 是为事件监视器指定的名称。

如果稍后决定要将 lock_participant_activities 逻辑数据组添加至事件监视器，那么可使用 ALTER EVENT MONITOR 语句：

```
ALTER EVENT MONITOR evmon-name ADD LOGICAL GROUP lock_participant_activities
```

此语句为新添加的逻辑数据组添加名为 LOCK_PARTICIPANT_ACTIVITIES_evmon-name 的表，另外还会修改事件监视器，以使事件监视器除了先前收集的数据之外，还收集来自 lock_participant_activities 逻辑数据组的数据。

限制：

ALTER EVENT MONITOR 语句只能用来将逻辑数据组添加至事件监视器。一旦添加逻辑数据组之后，就无法除去或删除逻辑数据组，也不能更改与满足以下条件的与表相关联的 PCTDEACTIVATE 的名称、目标表空间或值：用来捕获属于数据组的监视元素中数据的表。

13.6.7 其他监控增强

DB2 V10.1 还提供了如下管理增强：

1. 所有事件监视器的写至表支持

所有事件监视器都可将事件数据直接写至关系表。

2. 对先前发行版中创建的事件监视器输出表进行升级

使用新增的 EVMON_UPGRADE_TABLES 存储过程升级写至表和无格式事件(UE)表的事件监视器的现有目标表。

3. 修改未格式化的事件表中的数据

DB2 V10.1 对 EVMON_FORMAT_UE_TO_TABLES 存储过程添加 PRUNE_UE_TABLES 选项，从而可在将数据成功导出到关系表之后，将此数据从无格式事件(UE)表中删除。

4. 用于使你更详细地了解 DB2 服务器的新监视元素

- I/O 服务器的操作(预取程序)
- 应用程序提交的非嵌套活动的状态
- 有关 DATATAGINSC 阈值的信息
- 有关存储器组的信息
- 工作负载监视信息
- 连接和认证活动期间耗用的时间
- 有关程序包高速缓存中运行时间最长的 SQL 语句的详细信息
- 系统中耗用时间的其他度量方式
- 缓冲池和组缓冲池(对于 DB2 pureCluster 环境)活动
- 有关用法列表的信息
- 有关内存池和内存集用法的信息

13.7　金钟罩——安全功能增强

“金钟罩，金铸之铜覆罩全身”。数据安全可谓 DB2 的金钟罩、铁布衫。不安全的数据库是失败的，接下来让我们看看 DB2 V10.1 如何练就了这身神功。

DB2 V10.1 引入了行列访问控制(Row and Column Access Control，RCAC)，又给数据增加了一层安全保护。RCAC 可以控制表中的行、甚至是列可以被谁访问。

为了遵循企业、政府对数据安全的规定，可能需要在数据模型、程序逻辑方面花大量工夫来确保信息安全，必须做到只让需要的人访问需要的数据。而 RCAC 可以轻松帮我们做到这点。例如，在医疗系统中运行着 DB2，可以通过过滤病人的数据，而不让医生看到他们不必要知道的信息。又如，当病人服务代表去查询病人的信息表时，他们只能看到病人的姓名和电话这两列，而病人的治疗历史这一列对他们是隐藏的，可以显示为空，或者显示出其他非真实的内容。

13.7.1　RCAC 特点

- 没有哪个用户可以绕过 RCAC 的检查。即便具有 DATAACCESS 权限的用户也不能随心所欲地逃离 RCAC 的检查，只有 SECADM 才能管理 RCAC。因此，你可以通过 RCAC 来控制具有 DATAACCESS 权限的用户访问库里的所有表。
- RCAC 的检查是基于数据层面的，任何 SQL、应用、高级查询工具、报表工具都得遵循 RCAC 定义的规则。

- RCAC 的出现不会影响数据库的应用。RCAC 的中心思想是：把权限的关注点从对象转移到用户，即不再定义哪些对象可以被谁访问，而是哪些人可以访问哪些对象。而且即使 RCAC 隐藏了用户对表上某些列的访问，也不会向应用报出任何警告和错误，这就使 DBA 和程序设计者必须要小心，因为以前的查询不一定能看到表里的所有数据，必须显式赋予权限才行。

13.7.2 RCAC 规则

RCAC 从数据本身控制了表一级的访问、用户是否可以访问、基于用户的权限以及表上相关联的访问控制规则。RCAC 提供了行许可和列许可两种规则。

- 行许可权：行许可权是数据库对象，表示特定表的行访问控制规则。行访问控制规则是 SQL 搜索条件，描述用户可以访问的行集。
- 列掩码：列掩码是数据库对象，表示表中特定列的列访问控制规则。列访问控制规则是 SQL CASE 表达式，描述用户有权查看的列值以及要满足的条件。

可以使用 SQL 语句来定义，修改和删除 RCAC 的规则，也可以通过内置的 FUNCTION 来定义行和列的访问控制规则。听起来好像很多名词有点混乱，其实只要明确思路就理解了。行许可就相当于在 SQL 语句的 WHERE 后面加几个过滤条件；列掩码就相当于在某些列上做一些处理后再返给用户。以下举出两个示例来帮助大家理解行许可和列掩码的概念，更详细的内容请参考 DB2 信息中心。

13.7.3 RCAC 实战

行许可

比如有一张 STAFF 表，员工和经理都在里面，员工只能访问员工的信息，经理可以访问所有人的信息。我们需要先创建行许可对象，代码如下：

```
CREATE PERMISSION emp_row_acc ON db2inst1.staff
FOR ROWS WHERE
(VERIFY ROLE FOR USER(SESSION USER,'EMPLOYEE')=1 AND JOB<>'Mgr')-------A
OR VERIFY ROLE FOR USER(SESSION USER,'MANAGER')=1 ----------------------B
ENFORCED FOR ALL ACCESS
ENABLE;
```

大家可以看到，里面有两个过滤条件 A 和 B，VERIFY_ROLE_FOR_USER 是 DB2 的内置方法。VERIFY_ROLE_FOR_USER(SESSION_USER,'EMPLOYEE')的意思就是：如果当前用户属于角色 EMPLOYEE，那么返回 1；否则返回 0。那么以上代码的目的就很明显：如果用属于 EMPLOYEE 角色的用户去查询、修改这个表，WHERE 的后面会隐藏的加上

AND JOB<>'Mgr'这部分，相当于上面代码的 A 部分；如果当前用户属于角色 MANAGER，那么在查询、修改这个表时，不会在 WHERE 后面加上任何隐藏的过滤条件，即所有记录都可以查询到。因而可以说，行许可就是在 WHER 后面加隐藏的过滤条件。

行许可对象创建后别忘了激活：

```
ALTER TABLE DB2INST1.STAFF ACTIVATE ROW ACCESS CONTROL;
```

注意：

删除行许可对象并不会立即撤销这个表上的约束，必须取消激活这个表上的访问控制才可生效。

列掩码

还是 STAFF 这张表，现在需求变了一点，要求普通员工不能看到其他人的岗位信息，只有经理可以看到所有人的岗位信息，即 JOB 列是否可见。创建列掩码的代码如下：

```
CREATE MASK emp_col_acc ON db2inst1.staff
FOR COLUMN job RETURN
CASE WHEN VERIFY ROLE FOR USER(SESSION USER,'EMPLOYEE')=1 THEN 'NO PERMISSION'
WHEN VERIFY_ROLE_FOR_USER(SESSION_USER,'MANAGER')=1 THEN job END ENABLE;
```

这里和行许可的例子很相似，如果当前用户属于 EMPLOYEE 角色，那么 JOB 列将返回“NO PERMISSION”字符串，而属于 MANAGER 角色的用户会看到 JOB 列的真实信息。其实列掩码只相当于在 SELECT 或 UPDATE 后跟着的字段上加了一层处理而已。

行列访问控制(RCAC)对数据的访问控制很严格，任何用户都无法跳过 RCAC 的检查。有时候看不到表中的任何数据以及记录数，你可能认为这个表中没有数据，然后将其 DROP，但很有可能其中是有数据的，只是看不到而已。所以用的时候一定要注意访问权限的控制。

在权限管理的实现中，我们需要从可管理性、性能、应用的支持程度等多方面去评估，到底是使用视图、RCAC、LBAC 还是触发器来实现，只有综合考虑各个技术的优缺点，才能制定一套强大、健壮的安全体系。

13.8　本章小结

电影“2012”的末尾是经历了巨变的地球，是承载了人类文明的 5 艘大船，是对新时代的展望。这正如当今的数据库大环境，随着 Hadoop、NoSQL 等非关系型数据库的兴起，关系型数据库正在遭受莫大的考验，希望 DB2 V10.1 在经历了如此多的改变后，能够成为电影里的诺亚方舟，继续传承关系数据库的文明。